May 2000

Point-to-point connectivity between neuromorphic chips using address events

Kwabena A. Boahen
University of Pennsylvania, boahen@seas.upenn.edu

Copyright 2000 IEEE. Reprinted from *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Volume 47, Issue 5, May 2000, pages 416-434.
Publisher URL: http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=18224&puNumber=82

Point-to-point connectivity between neuromorphic chips using address events

Abstract
This paper discusses connectivity between neuromorphic chips, which use the timing of fixed-height fixed-width pulses to encode information. Address-events ($\log_2(N)$-bit packets that uniquely identify one of N neurons) are used to transmit these pulses in real time on a random-access time-multiplexed communication channel. Activity is assumed to consist of neuronal ensembles--spikes clustered in space *and* in time. This paper quantifies tradeoffs faced in allocating bandwidth, granting access, and queuing, as well as throughput requirements, and concludes that an arbitered channel design is the best choice.The arbitered channel is implemented with a formal design methodology for asynchronous digital VLSI CMOS systems, after introducing the reader to this top-down synthesis technique. Following the evolution of three generations of designs, it is shown how the overhead of arbitrating, and encoding and decoding, can be reduced in area (from N to $\sqrt{N}$) by organizing neurons into rows and columns, and reduced in time (from $\log_2(N)$ to 2) by exploiting locality in the arbiter tree and in the row–column architecture, and clustered activity. Throughput is boosted by pipelining and by reading spikes in parallel. Simple techniques that reduce crosstalk in these mixed analog–digital systems are described.

Keywords
asynchronous logic synthesis, interchip communication, spiking neurons, virtual wiring

Point-to-Point Connectivity Between Neuromorphic Chips Using Address Events

Kwabena A. Boahen

Invited Paper

***Abstract*—This paper discusses connectivity between neuromorphic chips, which use the timing of fixed-height fixed-width pulses to encode information. Address-events ($\log_2(N)$-bit packets that uniquely identify one of N neurons) are used to transmit these pulses in real time on a random-access time-multiplexed communication channel. Activity is assumed to consist of neuronal ensembles—spikes clustered in space *and* in time. This paper quantifies tradeoffs faced in allocating bandwidth, granting access, and queuing, as well as throughput requirements, and concludes that an arbitered channel design is the best choice.The arbitered channel is implemented with a formal design methodology for asynchronous digital VLSI CMOS systems, after introducing the reader to this top-down synthesis technique. Following the evolution of three generations of designs, it is shown how the overhead of arbitrating, and encoding and decoding, can be reduced in area (from N to $\sqrt{N}$) by organizing neurons into rows and columns, and reduced in time (from $\log_2(N)$ to 2) by exploiting locality in the arbiter tree and in the row–column architecture, and clustered activity. Throughput is boosted by pipelining and by reading spikes in parallel. Simple techniques that reduce crosstalk in these mixed analog–digital systems are described.**

***Index Terms*—Asynchronous logic synthesis, interchip communication, spiking neurons, virtual wiring.**

I. Connectivity in Neuromorphic Systems

ENGINEERS are far from matching either the efficacy of neural computation or the efficiency of neural coding. Computers use a million times more energy per operation than brains do [1]. Video cameras use a thousand times more bandwidth per bit of information than retinas do (see Section II-A). We cannot replace damaged parts of the nervous system because of these shortcomings. To match nature's computational performance and communication efficiency, we must co-optimize information processing *and* energy consumption.

A small but growing community of engineers is attempting to build autonomous sensorimotor systems that match the efficacy and efficiency of their biological counterparts by recreating the function and structure of neural systems in silicon. Taking a structure-to-function approach, these *neuromorphic systems* go beyond bio-inspiration [2], copying biological organization, as well as function [3]–[5].

Neuromorphic engineers are using garden-variety VLSI CMOS technology to achieve their goal [6]. This effort is facilitated by similarities between VLSI hardware and neural wetware. Both technologies:

1) provide millions of inexpensive, poorly-matched devices;
2) operate in the information-maximizing low-signal-to-noise/high-bandwidth regime.

It is also challenged by these fundamental differences:

1) fan-ins and fan-outs are about ten in VLSI circuits versus several thousand in neural circuits;
2) most digital VLSI circuits are synchronized by an external clock, whereas neurons use the degree of coincidence in their firing times to encode information.

Neuromorphic engineers have adopted time-division multiplexing to achieve massive connectivity, inspired by its success in telecommunications [7] and computer networks [8]. The number of layers and pins offered by commercial microfabrication and chip-packaging technologies are severely limited. Multiplexing leverages the five-decade difference in bandwidth between a neuron (hundreds of hertz) and a digital bus (tens of megahertz), enabling us to replace thousands of dedicated point-to-point connections with a handful of high-speed metal wires and thousands of switches (transistors). It pays off, because transistors occupy less area than wires and are becoming relatively more compact in deep submicron processes.

In adapting existing networking solutions, neuromorphic architects are challenged by huge differences between the requirements of computer networks and those of neuromorphic systems. Whereas computer networks connect thousands of computers at the building- or campus-level, neuromorphic systems need to connect millions of neurons at the chip- or circuit-board level. Hence, they must improve the efficiency of traditional computer communication architectures, and protocols, by several orders of magnitude.

Mahowald and Sivilotti proposed using an *address-event representation* to transmit pulses, or spikes, from an array of neurons on one chip to the corresponding location in an array on a second chip [9], [4], [10]. In their scheme, depicted in Fig. 1, an address-encoder generates a unique binary address for each

Manuscript received November 1999; revised December 1999. This work was supported by ONR, DARPA, the Beckman Foundation, California Institute of Technology's National Science Foundation (NSF) Engineering Research Center for Neuromorphic Systems, the California Trade and Commerce Agency, the Office of Strategic Technology, Penn's Schools of Engineering and Applied Sciences and the School of Medicine (through the IME), the NSF Knowledge and Distributed Intelligence Program, and the Whitaker Foundation.

The author was with Carver Mead's Lab, California Institute of Technology, Pasadena, CA 91125 USA. He is now with the Bioengineering Department, University of Pennsylvania, Philadelphia, PA 19104-6392 USA.

Publisher Item Identifier S 1057-7130(00)04206-3.

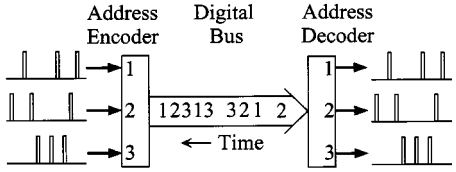


Fig. 1. The AER pulses from spiking neurons are transmitted serially by broadcasting addresses on a digital bus. Multiplexing is transparent if the encoding, transmission, and decoding processes cycle in less than Δ/n s, where Δ is the desired spike-timing precision and n is the maximum number of neurons that are active during this time (adapted from [4]).

neuron whenever it spikes. A bus transmits these addresses to the receiving chip, where an address decoder selects the corresponding location.

Eight years after Mahowald and Sivilotti proposed it, the address-event representation (AER) has emerged as the leading candidate for communication between neuromorphic chips. Indeed, at the National Science Foundation (NSF) Neuromorphic Engineering Workshop held in June/July 1997 at Telluride, CO, the AER Interchip Communication Workgroup was in the top two, second only to Mindless Robots in popularity [11].

The performance of the original point-to-point protocol has been greatly improved. Efficient hierarchical arbitration circuits have been developed to handle one-dimensional (1-D) and two-dimensional (2-D) arrays [12]–[14]. Sender and receiver interfaces have been combined on a single chip to build a transceiver [15]. Support for multiple senders and receivers [15]–[17], 1-D nearest-neighbor-connected network topologies [18], reprogrammable connections, and projective or receptive fields [19], [15], [17] has been added. Laboratory prototypes with 20 000 neurons and 120 000 AER-based connections have been demonstrated [19]. Systems with a million neurons and a billion connections are on the drawing board. In the near future, we are bound to see large-scale neuromorphic systems that rewire themselves—just like neural systems do—by taking advantage of the dynamically reprogrammable virtual wiring [20] made possible by AER.

In this paper, the goal is to provide a tutorial introduction to the design of AER-based interchip communication channels. The remainder of the paper is organized as follows. A simple model of neural population activity is introduced in Section II, which is used to quantify tradeoffs faced in communication channel design in Section III. This section is divided into four subsections that cover bandwidth allocation (Section III-A), channel access protocols (Section III-B), queuing (Section III-C), and throughput requirements (Section III-D). Having motivated an approach to inter-chip communication, the reader is introduced to a formal design methodology for asynchronous digital VLSI CMOS systems, and an AER communication channel implemented using this methodology is described in Section IV. This section is divided into four subsections that cover pipelining (Section IV-A), arbitration (Section IV-B), row–column organization (Section IV-C), and analog–digital interfaces (Section IV-D). The performance of three generations of designs is reviewed in Section V and the paper is summarized in Section VI. Parts of this work have been described previously in conference proceedings [21], [13], a magazine article [22], and a book chapter [14].

II. Neural Population Activity

Neuromorphic systems use the same currency of information exchange as the nervous system: fixed-height fixed-width pulses that encode information in their time of occurrence. Timing precision is measured by latency and temporal dispersion. *Neuronal latency* is the time interval between stimulus onset and spiking; it is inversely proportional to the strength of the stimulus. *Neuronal temporal dispersion* is due to variability between individual neurons; it is also inversely proportional to the strength of the stimulus. When messages are transmitted to reveal locations, or identities, of neurons that are spiking, the communication channel's finite latency and temporal dispersion add systematic and stochastic offsets to spike times that reduce timing precision.

Although a fairly general purpose implementation was sought, our primary motivation for developing a communication channel is to read spike trains off neuromorphic chips with thousands of spiking neurons, organized into 2-D arrays, such as silicon retinas [23], [24] or silicon cochlears [25], [26]. Neuronal activity is shaped by the preprocessing that occurs in the sensory epithelium, which is designed to eliminate redundancy and encode information efficiently [27], [28]. We optimized the channel design for the resulting neuronal population activity, and sought an efficient and robust implementation that supports adaptive pixel-parallel quantization. This design should be well suited to higher-level neuromorphic processors in so far as they code information efficiently.

A. Efficient Coding in the Retina

The retina converts spatiotemporal patterns of incident light into spike trains. Transmitted over the optic nerve, these discrete spikes are converted back into continuous signals by dendritic integration in postsynaptic targets. Retinal processing maximizes the information carried by these spikes. Sampling at the Nyquist rate, conventional imagers require 40 Gb/s to match the eyes' photopic range (17 bits), spatial resolution (60 cycles/°), temporal resolution (10 Hz), and field of view ($2 \times 90° \times 90°$). In contrast, coding 2 bits of information per spike [29], the million-axon optic nerve transmits just 40 Mb/s—a thousand times less.

The retina has evolved exquisitely adaptive filtering and sampling mechanisms to improve coding efficiency, six of which are highlighted below.

1) *Local automatic gain control* at the photoreceptor- [30] and network-level [24], [31] eliminates the dependence on lighting; the receptors respond to contrast instead. Adapting locally extends the retina's input dynamic range without increasing its output range.
2) *Bandpass spatiotemporal filtering* in the outer plexiform layer (or OPL, the retina's first stage) [24] passes an intermediate range of spatial frequencies *or* temporal frequencies. Rejecting low frequencies reduces redundancy, and rejecting high frequencies reduces noise.
3) *High-pass temporal and spatial filtering* in the inner plexiform layer (or IPL, the retina's second stage) [31] suppresses the OPL's strong low temporal-frequency response at its peak spatial frequency (i.e., sustained

response to static edge) and its strong low spatial-frequency response at its peak temporal frequency (i.e., blurring of moving edge).

4) *Half-wave rectification* in ON and OFF output cell types [31] eliminates the elevated neurotransmitter-release and spike-firing rates required to signal both positive and negative signal excursions using a single channel. ON/OFF encoding is used in bipolar cells (the OPL-to-IPL relay cells) as well as in ganglion cells (the retina's output cells).
5) *Phasic transient–sustained response* in the ganglion cells [32] avoids temporal aliasing by transmitting rapid transients using brief spike-bursts, and eliminates redundant sampling by transmitting slow fluctuations using a low sustained firing rate. Fig. 2 shows responses of silicon analogs of ganglion cells.
6) *Foveated architecture* and precise rapid eye movements provide the illusion of high spatial and temporal resolution everywhere, while sampling coarsely in time centrally and coarsely in space peripherally [33].

Since retinal neurons are driven by intermediate spatial and temporal frequencies, and are insensitive to low spatial and temporal frequencies, small subpopulations tend to fire together. Such correlated, but sparse, activity arises because the neurons respond to well-defined object features and adapt to the background. There is also evidence that gap-junction coupling between ganglion cells makes neighboring cells more likely to fire in synchrony [34], [35], and these coincident spikes drive downstream neurons more effectively [36]. The concept of a neuronal ensemble is introduced in the next section to capture this stimulus-driven fine spatiotemporal structure.

B. The Neuronal Ensemble

We can describe the activity of a neural population by an ordered list of locations in spacetime

$$\mathcal{E} = \{(x_0;\ t_0), (x_1;\ t_1), \cdots (x_i;\ t_i), \cdots\};$$
$$t_0 < t_1 < \cdots t_i < \cdots$$

where each coordinate specifies the occurence of a spike at a particular location, at a particular time. The same location can occur in the list several times, but a particular time can occur only once—assuming time is measured with infinite resolution.

There is no need to record time explicitly if the system that is logging this activity operates on it in real-time; only the location is recorded and time represents itself. In that case, the representation is simply

$$\mathcal{E} = \{x_0, x_1, \cdots x_i, \cdots\}; \qquad t_0 < t_1 < \cdots t_i < \cdots.$$

This real-time code is called the *address-event representation* (AER) [9], [10].

$\mathcal{E}$ has a great deal of underlying structure that arises from events occurring in the real world, to which the neurons are responding. The elements of $\mathcal{E}$ are clustered at temporal locations where these events occur, and are clustered at spatial locations determined by the stimulus pattern. Information about stimulus timing and pattern can therefore be obtained by extracting these clusters. Also, $\mathcal{E}$ has an unstructured component that arises from noise in the signal and in the system, and from differences in gain and state among the neurons. This stochastic component limits the precision with which the neurons can encode information about the stimulus. These statistically-defined clusters are called *neuronal ensembles*.

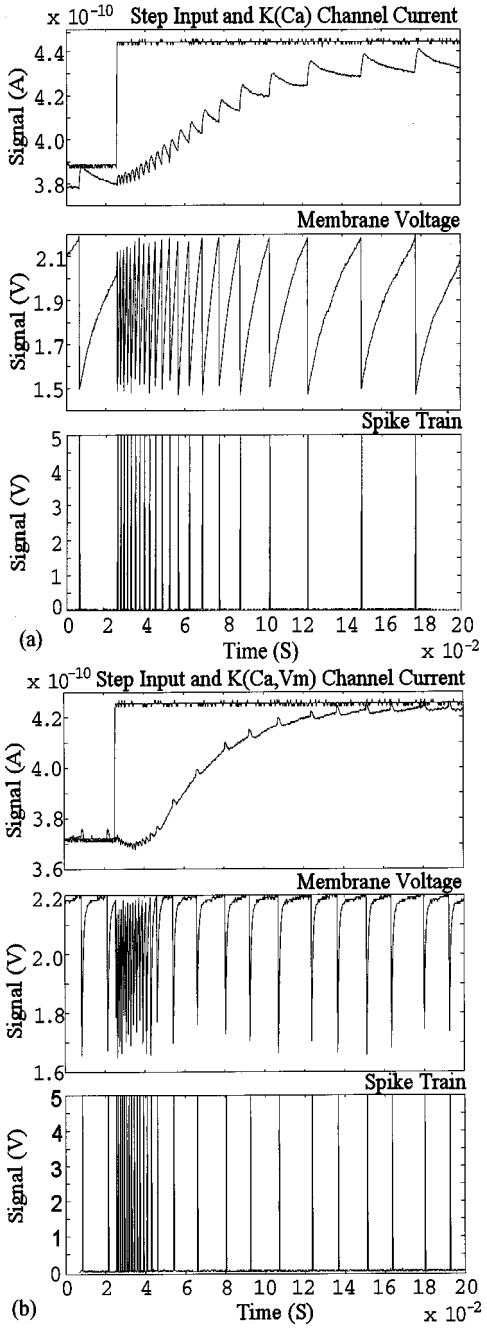


Fig. 2. Adaptive silicon neuron's step response. (a) Spike-frequency adaptation. Top: the integrator's output current builds up each time the neuron spikes, modeling calcium-dependent potassium channels. Middle: The membrane voltage charges from reset (1.5 V) to threshold (2.2 V), driven by the difference between input and integrator currents. Bottom: Spikes generated each time the membrane voltage reaches threshold. (b) Time-constant adaptation. The membrane voltage repolarizes rapidly because the integrator's output is temporarily shut off when the neuron is reset, modeling voltage-dependent potassium channels. Thus, a tight burst of spikes is generated and adaptation is rapid.

The probability distributions that describe these neuronal ensembles may be determined by characterizing a single neuron,

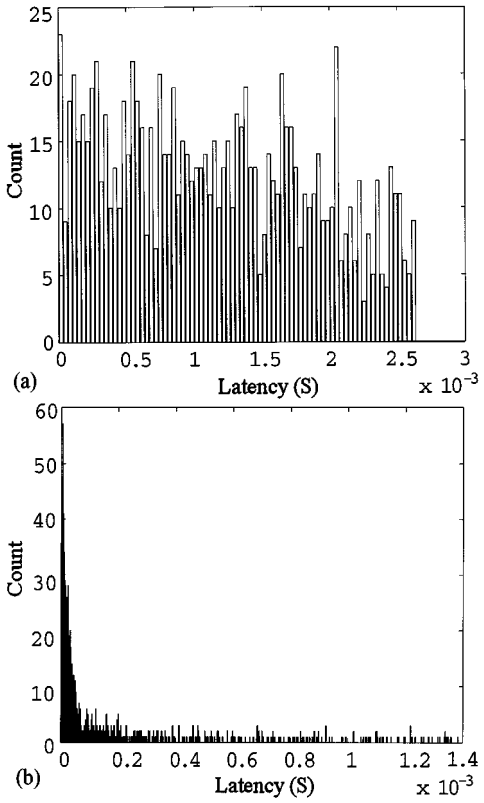


Fig. 3. Adaptive silicon neuron's latency distribution. The time taken to respond to a 15% step increase in input current was measured 1000 times. (a) Spike-frequency adaptation: the first spike is distributed more or less uniformly, with a slight tendency toward shorter latencies. The median is 1.3 ms and the firing rate immediately after the step (inferred from the longest latency of 2.63 ms) is 380 Hz, compared with a steady-state firing rate of 38.4 Hz. The bin size was 33.3 μs. (b) Time-constant adaptation: the distribution is heavily skewed toward shorter latencies. The median is 40 μs and the peak firing rate is 28.1 kHz, compared with a firing rate immediately after the step of 714.3 Hz and a steady-state firing rate of 62.5 Hz. The bin size was 2 μs.

assuming its state is randomized from trial to trial, just like the state is randomized across the population. It was measured how long it takes the adaptive silicon neuron to fire after a step change was made in its input current, repeated over several trials (described in [5], [32]). The results obtained with spike-frequency adaptation and time-constant adaptation, implemented by modeling calcium- and voltage-dependent ion channels in real neurons, are shown in Fig. 3.

The median of the distribution may serve as a measure of neuronal latency. It gives the expected latency if the target neuron's threshold equals 50% of the spikes in the ensemble. Unlike the simple integrate-and-fire neuron, whose latency is half its interspike interval, adaptive neurons have latencies that are much shorter than their steady-state interspike interval, as shown in Fig. 2. The ratio between the firing rate immediately after the step and the firing rate in steady state is defined as the *frequency adaptation*, γ [5]. The measurements shown in Fig. 2(a) and Fig. 3(a) yield $\gamma = 9.9$ for a silicon neuron that models calcium-dependent potassium channels. The ratio between the height of the peak in the distribution and the height of the uniform distribution over the same interval is defined as the *synchronicity*, ξ [5]. The measurements shown in Fig. 3(b) yield ξ =39.4 for a silicon neuron that models voltage-dependent potassium channels as well as calcium-dependent ones.

TABLE I
TIME-MULTIPLEXED COMMUNICATION CHANNEL DESIGN OPTIONS

Spec	Approaches	Remarks
Encoding	Amplitude	Long settling time, static power
	Width	Capacity $\propto^{-1}$ width
	Code	Inefficient for precision < 6bits
	Timing	**Minimum-width, rail-to-rail**
Latency	Polling	$\propto$ Number of neurons
	Event-driven	**$\propto$ Active fraction**
Integrity	Rejection	Collisions increase exponentially
	Arbitration	**Queue events**
Dispersion	Dumping	No waiting
	Queuing	**$\propto^{-1}$ surplus-capacity**
Capacity	Hard-wired	Simple $\Rightarrow$ Short cycle time
	Pipelined	**$\propto^{-1}$ slowest-stage**

In addition to characterizing the neuron's spike-timing precision relative to its steady-state firing rate, frequency adaptation and synchronicity allow us to compute its throughput requirements. Frequency adaptation gives the spike rate for neurons that are not part of the neuronal ensemble, assuming that these neurons have adapted. Synchronicity gives the peak spike rate for neurons in the ensemble. Throughput must exceed the sum total spike rate for these two segments of the population. A formula is derived for computing channel capacity requirements, as a function of tolerable percentage errors in spike rate and neuronal latency in the next section.

III. TRADEOFFS IN CHANNEL DESIGN

Four important performance criteria for a communication channel that provides virtual point-to-point connections between neuronal arrays are the following.

Capacity: The maximum rate at which spikes can be transmitted. It is equal to the reciprocal of the minimum communication-cycle time.

Latency: The median of the distribution of time intervals between spike generation in the sending population and spike reception in the receiving population.

Temporal Dispersion: The standard deviation of the latency distribution.

Integrity: The fraction of spikes that are delivered to the correct destination.

All four criteria together determine the *throughput*, which is defined as the usable fraction of the channel capacity. Because, the load offered to the channel must be reduced to achieve more stringent specifications for latency, temporal dispersion, and integrity.

Channel performance is affected by the information coding strategy used. Some alternatives to fixed-height fixed-width, pulses are listed in Table I, together with their pros and cons. The choices made in this work are set in boldface. Murray and Tarassenko explore the use of various pulse-stream representations to implement abstract models of neural networks [37], and Reyneri has analyzed and compared the performance of various pulse coding strategies [38]. However, little attention has been paid to using precise spike timing and neuronal ensembles

to encode information, despite increasing neurobiological evidence in support of such coding schemes [39], [40].

Given an information coding strategy, the communication channel designer faces several tradeoffs. Should he preallocate the channel capacity, giving a fixed amount to each user, or allocate capacity dynamically, matching each user's allocation to its current needs? Should she allow users to transmit at will, or implement elaborate mechanisms to regulate access to the channel? And how does the distribution of activity over time and over space impact these choices? Can he assume that users act randomly, or are there significant correlations between their activities? Light is shed on these questions in this section, and some definitive answers are provided.

A. Allocation: Dynamic or Static?

We may use adaptive neurons that sample at f_{Nyq} when the signal is changing, and sample at f_{Nyq}/Z when the signal is static, where Z is a prespecified attenuation factor. Let the probability that a given neuron samples at f_{Nyq} be a. That is, a is the *active fraction* of the population. Then, each quantizer generates bits at the rate

$$f_{\mathrm{bits}} = f_{\mathrm{Nyq}}(a + (1-a)/Z)\log_2 N$$

because a percent of the time, it samples at f_{Nyq}; the remaining $(1-a)$ percent of the time, it samples at f_{Nyq}/Z. Furthermore, $\log_2 N$ bits are used to encode the neuron's location, using AER, where N is the number of neurons.

On the other hand, we may use conventional quantizers that sample every location at f_{Nyq}, and do not locally adapt their sampling rate. In that case, there is no need to encode location explicitly. We simply poll all N locations, according to a fixed sequence, and infer the origin of each sample from its temporal location. As the sampling rate is constant, the bit-rate per quantizer is simply f_{Nyq}.

The multiple bits required to encode identity are offset by the reduced sampling rates produced by local adaptation when activity is sparse. In fact, adaptive sampling produces a lower bit rate than fixed sampling if

$$a < (Z/(Z-1))(1/\log_2 N - 1/Z).$$

For example, in a 64×64 array of neurons with sampling rate attenuation $Z = 40$, the active fraction a must be less than 6.1%.

It may be more important to minimize the number of samples produced per second—instead of minimizing the bit rate—as there are usually sufficient I/O pins to transmit all the address' bits in parallel. In that case, it is the number of samples per second that is fixed by the channel capacity. Given a certain fixed throughput F_{ch} in samples/s, we may compare the effective sampling rates f_{Nyq} achieved by various sampling strategies.

Adaptive neurons allocate channel throughput dynamically in the ratio $a : (1-a)/Z$ between active and passive fractions of the population. Hence

$$f_{\mathrm{Nyq}} = f_{\mathrm{ch}}/(a + (1-a)/Z) \tag{1}$$

where $f_{\mathrm{ch}} \equiv F_{\mathrm{ch}}/N$ is the throughput per neuron. The average neuronal ensemble size determines the active fraction a, and frequency adaptation and synchronicity determine the attenuation factor Z, assuming neurons that are not part of the ensemble have adapted. Fig. 4 shows how the sampling rate changes with the active fraction for various frequency adaptation factors $Z = \gamma$. For small a and $Z > 1/a$, the sampling rate may be increased by a factor of at least $1/2a$.

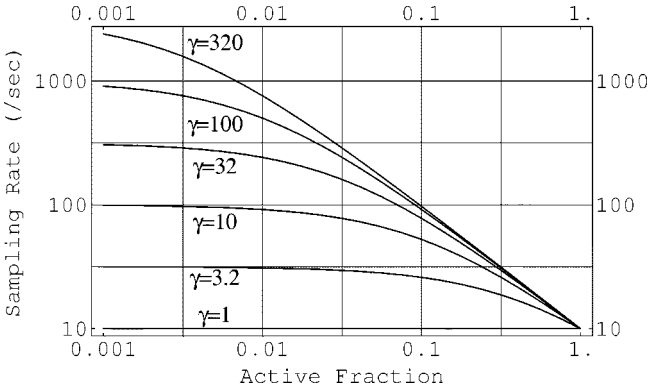


Fig. 4. Effective Nyquist sampling rate versus active fraction, plotted for various frequency adaptation factors (γ), with throughput fixed at 10 spikes/neuron/s. As the active fraction increases, the channel capacity must be shared by a larger number of neurons, and hence, the sampling rate decreases. It falls precipitously when the active fraction equals the reciprocal of the adaptation factor.

In a retinomorphic system, spatiotemporal bandpass filtering and half-wave rectification make output activity sparse [32], yielding active fractions of a few percent. Assuming $a = 0.05$, $Z = 1$ gives $f_{\mathrm{Nyq}} = f_{\mathrm{ch}}$ for the integrate-and-fire neuron; $Z = \gamma = 10$ gives $f_{\mathrm{Nyq}} = 6.9\ f_{\mathrm{ch}}$ for the neuron with frequency adaptation; and $Z = \gamma\xi = 450$ gives $f_{\mathrm{Nyq}} = 19.1\ f_{\mathrm{ch}}$ when the membrane time-constant adapts as well.

B. Access: Arbitered or Unfettered?

Assuming the spiking neurons are described by independent identically distributed Poisson point processes, the probability of k spikes being generated during a single communication cycle is given by

$$P(k, G) = G^k e^{-G}/k!$$

where G is the expected number of spikes. $G = T_{\mathrm{ch}}/T_{\mathrm{spk}}$, where T_{ch} is the cycle time and T_{spk} is the mean interval between spikes. By substituting $1/F_{\mathrm{ch}}$ for T_{ch}, where F_{ch} is the channel capacity and $1/(Nf_{\mathrm{nu}})$ for T_{spk}, where f_{nu} is the mean spike rate per neuron and N is the number of neurons, we find that $G = Nf_{\mathrm{nu}}/F_{\mathrm{ch}}$. Hence, G is equal to the offered load.

We may derive an expression for the collision probability, a well-known result from communications theory, using the probability distribution $P(k, G)$ [8]. To transmit a spike without a collision, the previous spike must occur at least T_{ch} seconds earlier, and the next spike must occur at least T_{ch} seconds latter. Hence, spikes are forbidden in a $2T_{\mathrm{ch}}$ time interval, centered around the time that transmission starts. Therefore, the probability of the spike making it through is $P(0, 2G) = e^{-2G}$, and the probability of a collision is

$$p_{\mathrm{col}} = 1 - P(0, 2G) = 1 - e^{-2G}.$$

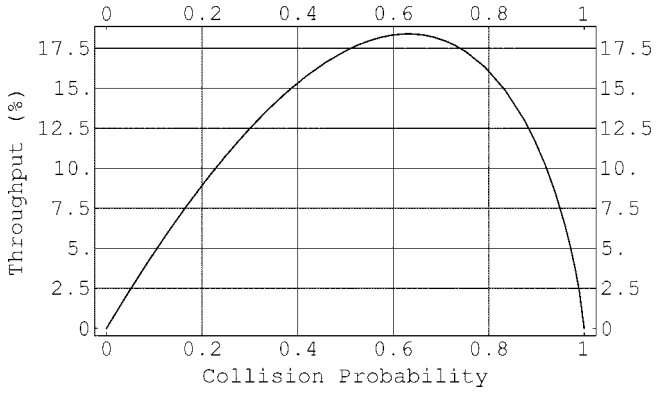


Fig. 5. Throughput versus collision probability. Throughput attains a maximum value of 18% when the collision probability is 0.64, and the load is 50%. Increasing the load beyond this level lowers throughput because collisions increase more rapidly than the load does.

The unfettered channel must operate at high error rates to maximize channel utilization. The throughput is $S = Ge^{-2G}$, since the probability of a successful transmission (i.e., no collision) is e^{-2G}. Throughput may be expressed in terms of the collision probability

$$S = \frac{1 - P_{\text{col}}}{2} \ln\left(\frac{1}{1 - P_{\text{col}}}\right) \tag{2}$$

This expression is plotted in Fig. 5. The collision probability exceeds 0.1 when throughput reaches 5.3%. Indeed, the unfettered channel utilizes a maximum of only 18% of its capacity. Therefore, it offers higher transmission rates than the arbitered channel only if it is more than five times faster, since, as we shall show next, the arbitered channel operates fine at 95% capacity.Contention occurs if two or more neurons attempt to transmit simultaneously when we provide random access to the shared communication channel. We can simply detect and discard samples corrupted by collision [41], or we can introduce an arbiter to resolve contention and a queue to hold waiting neurons [9], [10]. Unfettered access shortens the cycle time, but collisions increase rapidly as the load increases. Whereas arbitration lengthens the cycle time, reducing the channel capacity and queuing causes temporal dispersion, degrading timing information.

C. Latency: Queue New or Dump Old?

What about the timing errors introduced by queuing in the arbitered channel? For an offered load of 95 percent, the collision probability is 0.85. Hence, collisions occur frequently and neurons are most likely to spend some time in the queue. By expressing these timing errors as percentages of the neuronal latency and temporal dispersion, we can quantify the tradeoff between queuing new spikes, to avoid losing old spikes, versus dumping old spikes, to preserve the timing of new spikes.

To find the latency and temporal dispersion introduced by the queue, we use well-known results from queuing theory which give moments of the waiting time $\overline{w^n}$ as a function of moments of the service time $\overline{x^n}$ [42]

$$\overline{w} = \frac{\lambda \overline{w^2}}{2(1-G)}, \qquad \overline{w^2} = 2\overline{w}^2 = \frac{\lambda \overline{x^3}}{3(1-G)}$$

where λ is the arrival rate of spikes. These results hold when spikes arrive according to a Poisson process. With $x = T_{\text{ch}}$ and $\lambda = G/T_{\text{ch}}$, the mean and the variance of the cycles spent waiting are given by

$$\overline{m} \equiv \frac{\overline{w}}{T_{\text{ch}}} = \frac{G}{2(1-G)}, \qquad \sigma_m^2 \equiv \frac{\overline{w^2} - \overline{w}^2}{T_{\text{ch}}^2} = \overline{m}^2 + \frac{2}{3}\overline{m}.$$

We have assumed that the service time x always equals T_{ch}, and therefore $\overline{x^n} = T_{\text{ch}}^n$.

We find that at 95% capacity, for example, a sample spends 9.5 cycles in the queue, on average. This result agrees with intuition: As every 20th slot is empty, one must wait anywhere between 0–19 cycles to be serviced, which averages out to 9.5. Hence, the latency is 10.5 cycles, including the additional cycle required for service. The standard deviation is 9.8 cycles, virtually equal to the latency. In general, this is the case whenever the latency is much more than one cycle, resulting in a Poisson-like distribution for the wait times.

We can express the cycle-time T_{ch} in terms of the neuronal latency μ by assuming that T_{ch} is short enough to transmit half the spikes in an ensemble in that time. That is, if the ensemble has $N_{\mathcal{E}}$ spikes and its latency is μ, the cycle time must satisfy $\mu/T_{\text{ch}} = (N_{\mathcal{E}}/2)(1/G)$, since $1/G$ cycles are used to transmit each spike, on average, and half of them must be transmitted in μ seconds. Using this relationship, we can express the wait time as a fraction of the neuronal latency

$$e_\mu \equiv \frac{(\overline{m}+1)T_{\text{ch}}}{\mu} = \frac{G}{N_{\mathcal{E}}}\left(\frac{2-G}{1-G}\right).$$

The timing error is inversely proportional to the number of neurons because the channel capacity grows with population size. Therefore, the cycle time decreases, and there is a proportionate decrease in queuing time, even when the number of cycles spent queuing remains the same.

Conversely, given a timing-error specification, we can invert our result to find out how heavily we can load the channel. The throughput S will be equal to the offered load G, since every spike is transmitted eventually. Hence, the throughput is related to channel latency and population size by

$$S = N\left(\frac{e_\mu}{2} + \frac{1}{N_{\mathcal{E}}} - \sqrt{\left(\frac{e_\mu}{2}\right)^2 + \frac{1}{N_{\mathcal{E}}^2}}\right)$$

when the channel capacity grows linearly with the number of neurons. Fig. 6 shows how the throughput changes with the channel latency. It approaches 100% for large timing errors and drops precipitously for low timing errors, going below 95% when the normalized error becomes less than $20/N_{\mathcal{E}}$. As $N_{\mathcal{E}} = aN$, the error is $400/N$ if the active fraction a is 5%. Therefore, the arbitered channel can operate close to capacity with timing errors of a few percent when population size exceeds several tens of thousands.

D. Predicting Throughput Requirements

Given a neuron's firing rate immediately after a step change in its input f_a, we can calculate the peak spike rate of active neurons and add the firing rate of passive neurons to obtain the maximum spike rate. Active neurons fire at a peak rate of ξf_a,

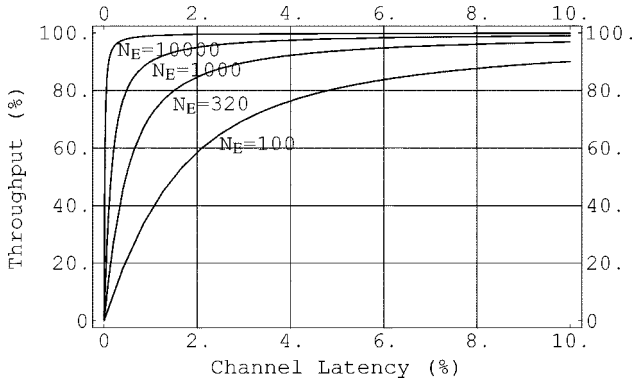


Fig. 6. Throughput versus normalized channel latency, plotted for different neuronal ensemble sizes ($N_{\mathcal{E}}$). Higher throughput is achieved at the expense of latency because queue occupancy goes up as the load increases. These wait cycles become a smaller fraction of the neuronal latency as the population size increases because cycle time decreases proportionately.

where ξ is the synchronicity, and passive neurons fire at f_a/γ (assuming they have adapted), where γ is the frequency adaptation. Hence, we have

$$F_{\max} = aN\xi f_a + (1-a)Nf_a/\gamma$$

where N is the total number of neurons and a is the active fraction of the population, which form a neuronal ensemble.

We can express the maximum spike rate in terms of the neuronal latency by assuming that spikes from the ensemble arrive at the peak rate. In this case, all aN neurons will spike in the time interval $1/(\xi f_a)$. Hence, the minimum latency is $\mu_{\min} = 1/(2\xi f_a)$. Thus, we can rewrite our expression for $F_{\max}$ as

$$F_{\max} = \frac{N}{2\mu_{\min}}\left(a + \frac{1-a}{\xi\gamma}\right).$$

Intuitively, $\mu_{\min}$ is the neurons' timing precision and $N(a+(1-a)/(\xi\gamma))/2$ is the number of neurons that fire during this time. The throughput must be equal to $F_{\max}$, and there must be some surplus capacity to minimize collision rates in the unfettered channel and minimize queuing time in the arbitered one. This overhead is over 455% [i.e., $(1-0.18)/0.18$)] for the unfettered channel, but only 5.3% [i.e., $(1-0.95)/0.95$)] for the arbitered one.

In summary, arbitration is the best choice for neuromorphic systems whose activity is sparse in space and in time, because we trade an exponential increase in collisions for a linear increase in temporal dispersion. Furthermore, holding utilization constant (i.e., throughput expressed as a percentage of the channel capacity), temporal dispersion decreases as technology advances and we build larger networks with shorter cycle times, even though the collision probability remains the same. The downside of arbitration is that it takes up area and time, reducing the number of neurons that can be integrated onto a chip and the maximum rate at which they can fire. Several effective strategies for reducing the overhead imposed by arbitration have been developed; they are the subject of the next section.

TABLE II
CHP LANGUAGE CONSTRUCTS

Operation	Notation	Explanation
Process	P_i	A composition of communications
Guard	G_i	$\equiv B_i \rightarrow P_i$. Execute P_i if B_i is true
Sequential	$P_1; P_2$	P_1 ends before P_2 starts
Overlapping	$P_1 \bullet P_2$	P_2 starts before P_1 ends, or vice versa
Concurrent	$P_1 \| P_2$	There is no constraint
Repetition	$*[P_1; P_2]$	$\equiv P_1; P_2; P_1; P_2; \ldots$. Repeats forever
Selection	$[G_1 \| G_2]$	Execute P_i for which B_i is true
Arbitration	$[G_1 \mid G_2]$	Required if B_i not mutually exclusive
Input	$A?x$	Read data from port A to register x
Output	$A!x$	Write data from register x to port A
Probe	$\overline{A}$	Is communication pending on port A?
Data type	$x : \mathsf{int}(m)$	Location x is an m-bit register
Field	$x.i$	Register x's ith bit
Assignment	$y := x$	Copy data from x to y

IV. ARBITERRED CHANNEL DESIGN

The design of arbitered channels that support point-to-point connections among spiking neurons on different chips is rather challenging. Early attempts were plagued by timing problems and crosstalk [9], [10]. Fortunately, significant progress has been made in asynchronous digital VLSI systems in recent years, culminating in the design of a microprocessor that uses no clocks whatsoever by Martin's group at Caltech [43]. We apply Martin's rigorous, correct-by-construction, design methodology to the arbitered channel, after introducing the program-based philosophy and notation it employs. Crosstalk, the pitfall of mixed analog–digital (MAD) system design, must also be addressed to achieve reliable and robust operation.

Martin's formal synthesis methodology enables us to design an asynchronous VLSI circuit by compiling a high-level specification, written in the *communicating hardware processes* (CHP) language, into a *production rule set* (PRS) [45]–[47]. A production rule evaluates a boolean expression in real time, and sets or clears a bit when the expression becomes true; it is straightforward to implement with MOS transistors. The synthesis procedure involves two intermediate steps: program decomposition and handshaking expansion.

Through *program decomposition* (PD), which involves decomposing the high-level specification into concurrent subprocesses, we:

1) reduce logical complexity by divide-to-conquer;
2) share expensive hardware resources.

At this level, we make architectural design decisions that simplify the design and minimize its hardware requirements. We must synchronize concurrent subprocesses and resolve contention for shared resources.

Ports are used to input data, to output data, or simply to synchronize, given that processes communicate when they reach particular points in their programs. Communication is described in CHP simply by writing down the name of the port, say S. This action may be composed with other communications using the language constructs outlined in Table II. A pair of complementary ports, one *active* and the other *passive*, are connected to form a channel, as shown in Fig. 7(a). Apart from complementary channel assignments, the only constraint on whether a port can be active or passive is the *probe*. A primitive operation, denoted $\overline{S}$, which a process invokes to check if a communication

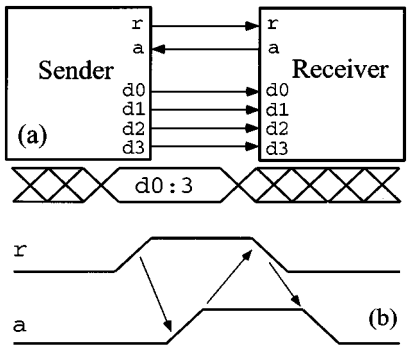


Fig. 7. Communication channel signals and timing. (a) Data-bus (d0, ···, d3) and handshake signals (r and a). (b) Timing diagram: the sender initiates the sequence by driving its data onto the bus and taking r high. The receiver acknowledges by taking a high, after latching the data. These two parties perform complementary sequences of actions and waits: r+; [a]; r−[ā] for the *active* sender and [r]; a+; [r̄]; a− for the *passive* receiver. The active port drives the so-called *request* line while the passive one drives the so-called *acknowledge* line.

TABLE III
HSE PRIMITIVES

Operation	Notation	Explanation
Signal	v	Voltage on a node
Complement	~v	Inversion of v
And	v & w	High if both are high
Or	v | w	Low if both are low
Set	v+	Drive v high
Clear	v-	Drive v low
Wait	[v]	Wait till v is high
Sequential	[u];v+	≡ u -> v+ in PRS
Concurrent	v+,w+	≡ v+,w+ in PRS
Repetition	*[...]	Just like in CHP

is pending on its port S. It returns true if there is one and false otherwise. The probe only works with a passive port, due to implementation constraints.

Through *hand-shaking expansion* (HSE), which involves fleshing out each communication into a full four-phase handshake cycle, we:

1) choose whether to make a port active or passive;
2) reshuffle a communication cycle's four phases.

At this level, we make logic design decisions that reduce memory and improve speed. PD and HSE produce sequences of waits and actions that define concurrent subprocesses. These sequences are converted into PRS by writing a rule to perform each action when the preceding wait becomes true.

The *four-phase handshake* is performed with a pair of wires, as shown in Fig. 7(b), and specified using the HSE primitives described in Table III. The active port initiates the handshake by asserting the so-called request signal (i.e., r+). The probe is implemented by monitoring this signal (i.e., [r]), an opportunistic implementation that works only with a passive port. Data is assumed to be valid when the request signal arrives, which requires their propagation delays to be matched. The matched delays required by this *bundled-data* protocol [47] can be avoided by using a *dual-rail* representation, but this delay-insensitive scheme requires two lines per data bit [45].

We can describe an AER transmitter as follows. The format used to specify a process in CHP is

$$\begin{aligned}\text{name(arguments)} \equiv\ & \underline{\text{process}}\text{(ports)}\\ & \text{program}\\ & \underline{\text{end}}\end{aligned}$$

We name the transmitter process $\mathsf{AEXMT}(N)$, giving the number of neurons N as an argument, and assign it N dataless ports, named L_n, to service N neurons, and a single output port, named A, that writes (represented by !) a $\lceil \log_2(N) \rceil$-bit integer.[1] All this information is specified in the header

$$\begin{aligned}\mathsf{AEXMT}(N) \equiv\ & \underline{\text{process}}(L_1, L_2, \cdots, L_N, A!\text{int}\\ & (\lceil \log_2(N) \rceil))\\ & \text{program}\\ & \underline{\text{end}}\end{aligned}$$

Now, we write a program that probes the L-ports to detect communications initiated by neurons that are spiking, and arbitrates (represented by |) between them. It then communicates on the chosen port and transmits its address; these operations may occur concurrently (represented by ||). The code for this algorithm is

$$*[[\overline{L_1} \rightarrow A!\text{enc}(1) || L_1 | \cdots | \overline{L_N} \rightarrow A!\text{enc}(N) || L_N]].$$

A function $\mathtt{enc}(n)$, which converts a *one-hot* code into a binary one, is invoked to encode the chosen port's address. The inner brackets delimit arbitration while the outer ones delimit repetition, together with the asterix.

Similarly, we can describe an AER receiver in CHP as follows. The receiver uses a $\lceil \log_2(N) \rceil$-bit input port, named A, to read (represented by "?") address-events and uses N dataless ports, named R_n, to service N neurons. Thus, we have

$$\begin{aligned}&\mathsf{AERCV}(N)\\ &\quad \equiv \underline{\text{process}}(A?\text{int}(\lceil \log_2(N) \rceil), R_1, R_2, \cdots, R_N)\\ &\qquad b\text{: int}(\lceil \log_2(N) \rceil)\\ &\qquad c\text{: int}(N)\\ &\qquad *[A?b;\ c9:\ = \text{dec}(b);\ [c.1 \rightarrow R_1 || \cdots || c.N \rightarrow R_N]]\\ &\qquad \underline{\text{end}}\end{aligned}$$

A function $\mathsf{dec}(b)$, which converts from binary to one-hot, is invoked to decode the address. b and c are local $\lceil \log_2(N) \rceil$-bit and N-bit registers, respectively, used to store the input and the result. The receiver communicates on the port corresponding to the one set bit $c.i$ in the one-hot code c. This port is chosen by *selection* (represented by ||), which is used when there is no need for arbitration (i.e., the choice is unique). The inner brackets delimit selection while the outer ones delimit repetition.

A. Pipelining

Pipelining, a well-known approach to increasing throughput, reduces the time-overhead of arbitration by breaking the communication cycle up into a sequence of smaller steps that ex-

[1] $\lceil x \rceil$ gives the smallest integer larger than, or equal to, x.

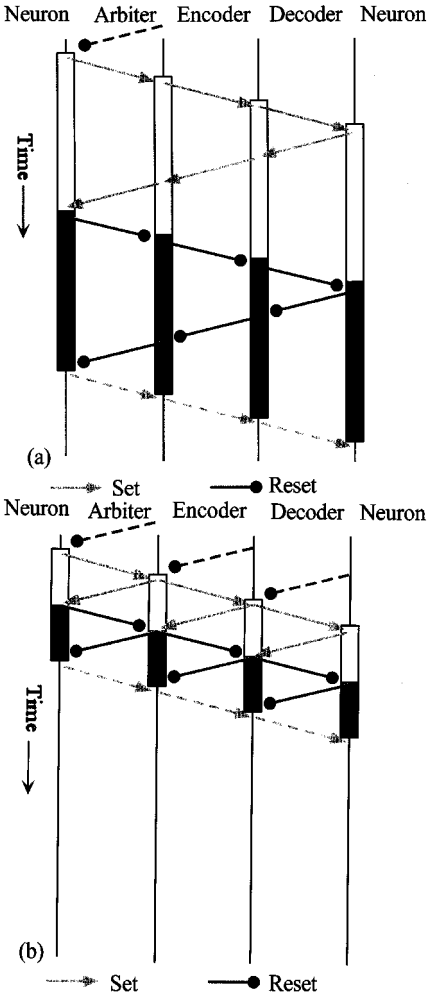


Fig. 8. Pipelined communication cycle. (a) Communication cycle involving four-phase handshakes between sending neuron, arbiter, address encoder, address decoder, and receiving neuron. White and black boxes indicate the duration of the set and reset halves. Preceeding or succeeding cycles are in dashed lines. (b) In the pipelined channel, we do not wait for the next stage to acknowledge us before we acknowledge the previous stage. Similarly, we do not wait for it to withdraw its acknowledge before we withdraw ours.

ecute concurrently. Concurrency reduces the cycle-time to the length of the longest step, with several address-events in various stages of transmission at the same time, as shown in Fig. 8. Handshaking makes pipelining, and queuing, straightforward: you can stall a pipeline stage, or make a neuron wait, simply by refusing to acknowledge it. To become conversant with the synthesis procedure, let us design a handshake circuit to coordinate the request and acknowledge signals of adjacent stages in a pipeline and control data transfer.

A data-buffer pipeline stage (also called a FIFO, for first-in first-out) is described in CHP as

$$\begin{aligned}\mathsf{LRBUF} \equiv\ & \underline{\mathsf{process}}(L?\mathsf{int}(4),\ R!\mathsf{int}(4)) \\ & b\colon \mathsf{int}(4) \\ & *[L?b;\ R!b] \\ & \underline{\mathsf{end}}\end{aligned}$$

It reads in a nibble from its L port, turns around, and writes out the nibble on its R port. We make L passive and R active, which

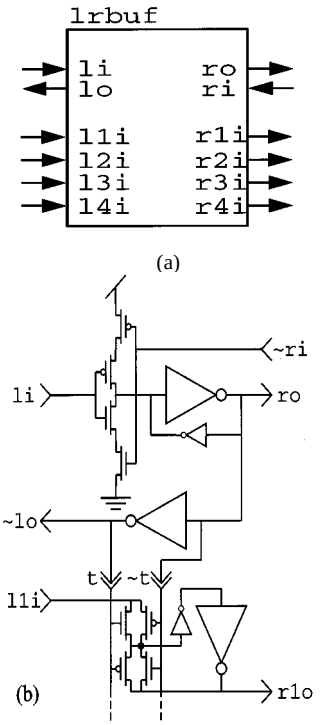


Fig. 9. Data-buffer pipeline stage. (a) HSE description: each port is fleshed out into a pair of handshake lines and a set of data lines, and assigned an active or passive role. (b) Circuit description: the data-buffer consists of a latch and a *C-element*—a gate whose output goes high when both inputs are high and goes low when both inputs are low. As its output is not always actively driven, a weak feedback inverter, called a *staticizer*, is added to hold state.

allows buffers to be cascaded, and label these ports' request and acknowledge signals as shown in Fig. 9(a). Thus, we obtain the following HSE:

$$*[[\mathtt{li}];\ \mathtt{lo}+;\ [\tilde{\mathtt{li}}];\ \mathtt{lo}-;\ \mathtt{ro}+;\ [\mathtt{ri}];\ \mathtt{ro}-;\ [\check{\mathtt{ri}}]]$$

simply by replacing each communication with a full four-phase handshaking sequence.

In the pipelined example shown in Fig. 8(b), the first half of R occurs after the first half of L; the second half of R remains at the end. That is

$$*[[\check{\mathtt{ri}}\ \&\ \mathtt{li}];\ \mathtt{lo}+,\ \mathtt{ro}+;\ [\mathtt{ri}\ \&\ \tilde{\mathtt{li}}];\ \mathtt{lo}-,\ \mathtt{ro}-;]$$

$[\check{\mathtt{ri}}]$ has been postponed to the beginning of the next cycle, making R *lazy-active*, and adjacent waits and adjacent actions have been merged. This reshuffling is logically equivalent to our original CHP program (i.e., $L;\ R$), because both data exchange and synchronization occur during the communication's first half (the second half conveniently returns the signals to their original state).

The pipelined sequence operates as follows. When data arrives (i.e., $[\mathtt{li}]$), we latch it and acknowledge receipt ($\mathtt{lo}+$). However, we wait until the next stage has transmitted the previous item ($[\check{\mathtt{ri}}]$) so that we can pass on the new data ($\mathtt{ro}+$) at the same time. We must keep data available until we get an acknowledge ($[\mathtt{ri}]$). Then it is safe to make the latch transparent again and to withdraw our request ($\mathtt{ro}-$). However, we wait for the previous stage to withdraw its request ($[\tilde{\mathtt{li}}]$) so that we can withdraw our acknowledge ($\mathtt{lo}-$) at the same time.

We implement the pipelined sequence by writing production rules that perform actions in the sequence when preceding waits become true

$$\check{\texttt{ri}} \,\&\, \texttt{li} \rightarrow \texttt{lo}+,\ \texttt{ro}+,\ \texttt{t}-$$
$$\texttt{ri} \,\&\, \tilde{\texttt{li}} \rightarrow \texttt{lo}-,\ \texttt{ro}-,\ \texttt{t}+.$$

A strobe signal t has been added, which makes the latch opaque when low, and transparent when high. Sometimes we have to *strengthen* guards to prevent rules from misfiring or interfering [45], [47], but this is not required here.

The handshake logic for our passive–active data buffer is realized by a single gate, which acknowledges the previous stage, sends a request to the next stage, and strobes the latch, as shown in Fig. 9(b). The pull-down implements the first rule and the pull-up implements the second one. Setup and hold times may be satisfied by delaying the request, relative to the data, and by delaying withdrawing the data after an acknowledge is received [44].

In the unpipelined example shown in Fig. 8(a), on the other hand, the first half of R occurs immediately after [li]. That is

$$*[[\texttt{li}];\ \texttt{ro}+;\ [\texttt{ri}];\texttt{lo}+;\ [\tilde{\texttt{li}}];\ \texttt{ro}-;\ [\check{\texttt{ri}}];\ \texttt{lo}-].$$

Communications intertwined in this way are specified by the bullet (i.e., $L \bullet R$) in CHP. This HSE is implemented by the following PRS:

$$\texttt{li} \rightarrow \texttt{ro}+ \quad \texttt{ri} \rightarrow \texttt{lo}+$$
$$\tilde{\texttt{li}} \rightarrow \texttt{ro}- \quad \check{\texttt{ri}} \rightarrow \texttt{lo}-.$$

A pair of wires, connecting li to ro and ri to lo, suffices! We have to give up this simplicity to gain the speed-up offered by pipelining. Additional speed improvements may be made by exploiting locality in the arbiter and in the array, as shown in the next two subsections.

B. Arbitration

Arbitration may be performed by a recursive procedure.

1) Divide the neurons into two groups.
2) Choose one group, making sure there is an active neuron in the group you choose.
3) If the chosen subgroup has more than one neuron, repeat Steps 1 and 2 with this group.
4) Else, you are done.

Dividing by two balances the sizes of the subgroups, giving neurons in each subgroup equal chances of being picked.

In CHP, our recursive arbitration procedure is described by the recursive equation

$$\mathsf{ARB}(X) \equiv \mathsf{ARB}(X/2)||\mathsf{ARB}(2)||\mathsf{ARB}(X - X/2)$$

where $\mathsf{ARB}(X)$ is an X-input arbiter process, which consists of three subprocesses that run concurrently. These subprocesses are connected in a tree-like structure, as shown in Fig. 10(a). The recursion unwinds at $\mathsf{ARB}(2)$ or $\mathsf{ARB}(1)$, and hence, we only need to design a two-input arbiter cell; the one-input case is trivial. $(N-1)$ $\mathsf{ARB}(2)$ cells, connected in a balanced binary tree with $\lfloor \log_2(N) \rfloor$ levels, are needed to arbitrate between N neurons.[2]

[2] $\lfloor x \rfloor$ gives the largest integer smaller than x.

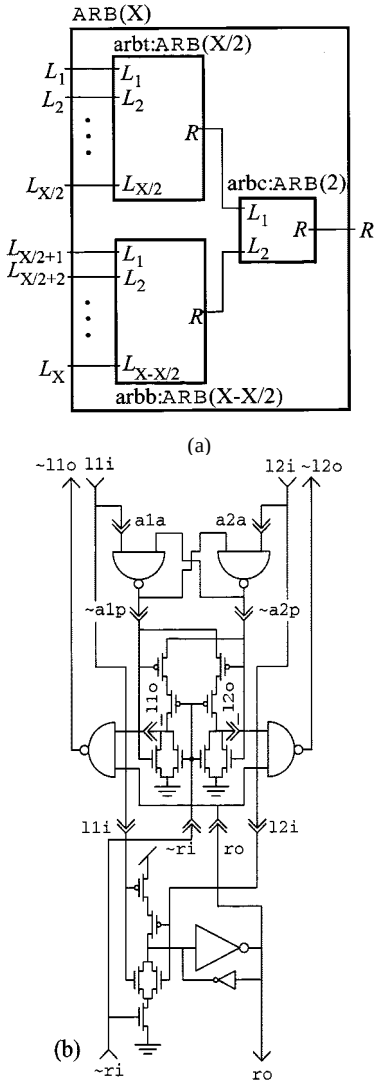


Fig. 10. Recursively-defined arbiter. (a) An X-input arbiter is built from a $X/2$-input arbiter (arbt), a $(X - X/2)$-input arbiter (arbb), and a two-input arbiter (arbc), connected as shown. The $X/2-$ and $(X - X/2)$-input arbiters are themselves recursively defined by the same procedure. (b) Greedy two-input arbiter vircuit. Requests, l1i and l2i, propagate down through a modified OR gate (bottom), while acknowledges, ri, propagate up through a router (middle). A flip-flop (top) arbitrates between the requests and controls the router, which steers the active-low acknowledge to the chosen request by NORing it with the flip-flops's active-low outputs (the source-switched pFET's filter metastable oscillations). A pair of NAND gates invert active-high acknowledges from the steering circuit and blocks them when the outgoing active-high request ro is low.

In CHP, the two-input arbiter cell is described by

$$\mathsf{ARB}(2) \equiv \underline{\mathsf{process}}(L_1,\ L_2,\ R)$$
$$*\,[[\overline{L_1} \rightarrow R;\ L_1;\ L_1;\ R|\overline{L_2} \rightarrow R;\ L_2;\ L_2;\ R]]$$
$$\underline{\mathsf{end}}$$

This process probes its L ports to determine if there are active neurons in either subgroup. Next, it communicates on its R port to ensure that the group of neurons it serves has been chosen. And finally, it communicates on either of its L ports to select an active subgroup. Thus, requests are relayed up the tree by probing the R-to-L channels, while selection is relayed down

the tree by communicating on the same channels. A second pair of L and R communications terminates the selection. The cell at the top of the tree, which serves all N neurons, is special, since its group is always chosen. Thus, communications on its R port are superfluous. We can connect its R port to a process that automatically completes the communication (i.e., $*[L]$).

Making L_1 and L_2 passive, and R active, the synthesis procedure yielded the circuit shown in Fig. 10(b). All paired communications were implemented using two halves of a single four-phase communication. As is normally done. A flip-flop (i.e., cross-coupled NAND-gates) was used to guarantee *mutual exclusion* by ANDing one port's active-high request with the other's active-low acknowledge [45].

The reshuffling implemented works as follows. When a request is received from the lower level (i.e., [l1i]), we send it to the flip-flop (a1a+) and, without waiting for a decision, we also relay it to the upper level (ro+). But we make sure the upper level has cleared its acknowledge to the previous request first ([ri]). If not, we do not make a new request. Instead, we accept the old acknowledge, assuming it is stable (ro is high), and relay it to the lower level (l1o+) as if it was a new acknowledge, once the arbitration subprocess ([a1p]) acknowledges.

At this point, we are half way through the communication cycle, and every signal is activated. When the lower level clears its request (i.e., [$\tilde{\text{l1i}}$]), we clear our request to the flip-flop (a1a−). We wait for its acknowledge to clear ([$\tilde{\text{a1p}}$]) before we clear our acknowledge to lower level (l1o−), preventing a new incoming request from using an unstable $\tilde{\text{a1p}}$ signal. However, we clear our request to the upper level (ro−) only if both incoming requests have been cleared. A strategy that allows our sister process to service her daughters with the old acknowledge.

The modified OR-gate's staticizer and the pull-ups of the flip-flop's NAND gates must not be too weak. Otherwise, slow transitions on the incoming request lines (i.e., l1i or l2i) make the modified OR gate's output oscillate [refer to Fig. 10(b)]. This happens when the incoming acknowledge ($\check{\text{ri}}$) arrives before the outgoing request signal (ro) completes its transition,[3] because the pull-up overcomes the staticizer when the active-low acknowledge disables the pull-down. In practice, this occurs only at the top cell, where the outgoing request is immediately fed back through an inverter. And, if the pull-ups in the flip-flop's NAND gates are too weak, the router circuit loads the flip-flop, pulling the higher output down. Thus, it reduces the differential signal, causes both signals to creep downward, and produces nonmonotonic transitions.

Additionally, two conditions must be met to prevent a lingering acknowledge from the flip-flop from servicing a new incoming request [refer to Fig. 10(b)].

1) $\tilde{\text{a1p}}$, not $\check{\text{ri}}$, fires _l1o−. Hence, when _l1o goes low, we know that $\tilde{\text{a1p}}$ is high. This condition is easily satisfied, as the number of gates in these two paths differs a lot. The downward transition on l1i propagates through the flip-flop to drive $\tilde{\text{a1p}}$ high, but propagates through the modified OR-gate and three gates at the next level to drive $\check{\text{ri}}$ high.

[3]Tim Horiuchi discovered this instability.

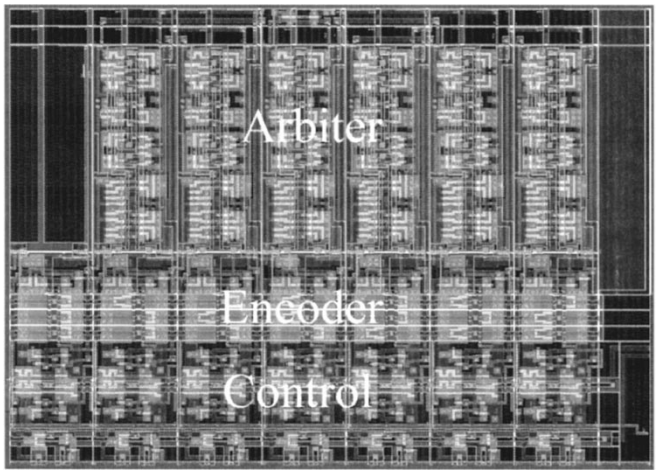


Fig. 11. Layout of recursively-defined arbiter. A seven-input arbiter, with address encoder and control cells. It is built up from a three-input arbiter (first two cells) and a four-input arbiter (last three cells), which are connected together by an additional cell (third cell), for a total of six cells. These three- and four-input arbiters are built from two- and one-input arbiters; the latter is just a pair of wires that bypass the lowest level of the tree. An inverter at the top ties the active-high outgoing request back to the active-low incoming acknowledge. Wells and selects have been omitted for clarity. Gray is substrate; black and darker shading is M2. The pitch is 70λ and the height is 380λ, or 21 μm × 114 μm in 0.5-μm technology ($\lambda = 0.3$ μm).

2) $\check{\text{\_l1o}}$, not ro, fires $\tilde{\text{l1o}}+$. Hence, when $\tilde{\text{l1o}}$ goes high, we know that _l1o is low. This condition requires careful transistor sizing, as the number of gates is these two paths are identical. The downward transition on l1i propagates through the flip-flop and the NOR gate to drive _l1o low, but propagates through the modified OR gate, and its inverter, to drive ro low.

Layout for the arbiter tree is shown in Fig. 11. This layout was generated by implementing the recursive algorithm in a silicon compiler program, starting with layouts for the two-input arbiter cell. The program was written in C using the layout-editor's (L-Edit) user-programmable interface (UPI) and layout-compilation libraries (L-Comp), all from Tanner Research, Inc. We now turn our attention to reducing the area-overhead of arbitration by tiling neurons in 2-D arrays.

C. Row–Column Organization

By going to a hierarchical X-column–Y-row organization, as proposed in [9], [10], we reduce the number of two-input arbiter cells from $Y \times X - 1$ to $Y + X - 2$. That is, it cost us nothing for the first row or column and one arbiter cell for each additional row or column. Hence, the area-overhead scales like $\sqrt{N}$, where N is the number of neurons. The number of address-encoder and decoder cells are also reduced by a similar amount: one per row or column, instead of one per neuron. Both sending and receiving neural populations may be organized into 2-D arrays.

1) 2-D Transmitter: Neurons in a 2-D AER transmitter are selected by performing hierarchical row-first column-second arbitration, as shown in Fig. 12. First, we use a Y-input arbiter to choose one of Y rows, and then we use a X-input arbiter to choose one of X neurons assigned to that row. Hierarchical arbitration guarantees that only one row is active at any time.

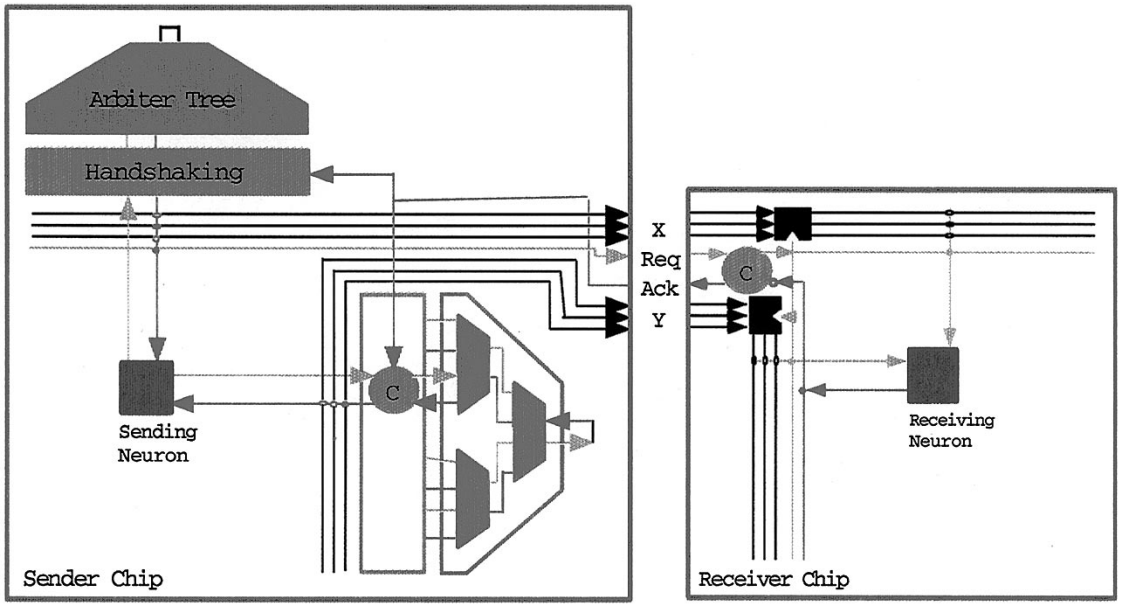


Fig. 12. Architecture of address-event transmitter and receiver. The sending neuron's interface circuit [shown in [Fig. 13(a)] communicates spikes to peripheral circuitry using row and column request–select lines. A row–column controller [Fig. 13(b)] relays requests from a row or column of neurons to the arbiters and relays the arbiter's acknowledge back; it also activates the address-encoder [Fig. 15(a)]. On the receiving end, a pipeline stage [Fig. 9(b)] reads and latches the address (X and Y). It acknowledges receipt right way and activates the address decoders [Fig. 15(b)], which select the corresponding row and column. The receiving neuron [Fig. 14] sends an acknowledge when both its row and column are selected. This signal is relayed to the pipeline stage by a two-level wired-OR circuit.

Hence, we can share a single X-input column-arbiter between all the rows. We must OR together all requests within each row to generate requests for the row arbiter, and all requests within each column to generate requests for the column arbiter. We save time by servicing all active neurons in the chosen row before we pick another row [13], [14]. However, we should not wait for its inactive neurons to communicate on the column lines. Only neurons that were active *at the time that the row was selected* must be serviced. This way, inactive neurons cannot prolong completion indefinitely if they subsequently spike.

This strategy is realized by the neuron-interface and row–column-control circuits shown in Fig. 13, designed by decomposing AEXMT(N) into row and column subprocesses, and following the synthesis procedure. The neuron drives the row-request line low (i.e., $\tilde{\mathtt{p}}-$) when a spike occurs ([lix]). The controller relays this request to the row arbiter (ro+) and grants the request by driving the row-select line high (s+) when the arbiter acknowledges ([ri]). It also activates the row-address–encoder (ao+). If necessary, the controller waits until previous column and encoder communications are completed ([$\tilde{\mathtt{a}}$i]). When the row is selected, all neurons with spikes place requests on their column lines ($\check{\mathtt{c}}$ox−) and clear their spikes one by one, by taking $\tilde{\mathtt{l}}$ox low, as these requests are granted ([cix]).

Each neuron releases the row- and column-request lines when it is serviced. The row-request line $\tilde{\mathtt{p}}$ goes high only when all the spikes have been cleared. The row-controller then withdraws its request to the arbiter (ro−), but it waits until it receives an acknowledge from the encoder ([ai]), since this signal prevents interference with ongoing communications. As soon as the arbiter clears its acknowledge ([$\tilde{\mathtt{r}}$i]), the controller withdraws its request to the encoder (ao−) and deselects the row (s−). The guard of $\tilde{\mathtt{l}}$ox− was stregthened to ensure that a neuron is reset only when its row *and* its column are selected.

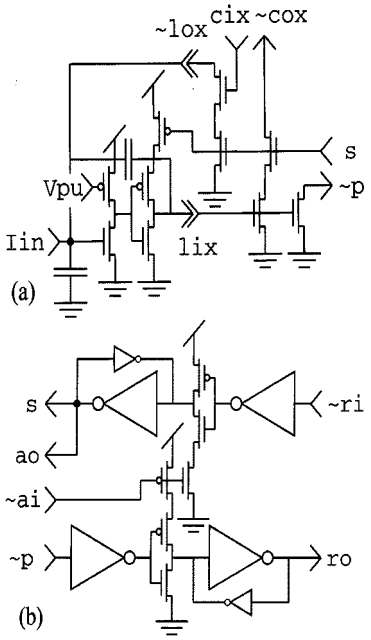


Fig. 13. Sending-neuron and row–column control circuits. (a) Five-transistor interface (right half) between neuron (i.e., lix, $\bar{\mathtt{l}}$ox) and row- ($\bar{\mathtt{p}}$, s) or column- ($\bar{\mathtt{c}}$ox, cix) control circuits. The pull-downs on $\bar{\mathtt{p}}$ and $\bar{\mathtt{c}}$ox form row- and column-wide wired-NOR gates; current source pull-ups are at the edge of the array. The neuron is disabled when its row is selected (s is high) to prevent generation of new spikes. Capacitive positive-feedback in the axon-hillock circuit (left half) provides hysteresis and speeds up transitions. (b) Interface among row or column (i.e., $\bar{\mathtt{p}}$, s), arbiter (ro, $\bar{\mathtt{r}}$i), and address-encoder (ao, $\bar{\mathtt{a}}$i). These gates are called *aC-elements* (for asymmetric): their outputs are set when both inputs are high and cleared when a particular input is low (or vice versa).

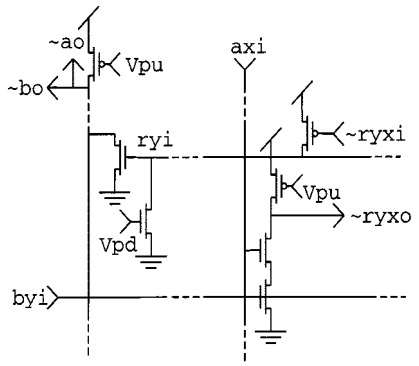


Fig. 14. Receiver's neuron interface circuit. Active-high column- and row-select signals axi and byi are NANDed together to generate an active-low request ṙyxo. The neuron responds with an active-low acknowledge ṙyxi. If desired, ṙyxo may be tied directly to ṙyxi to produce a minimum-width active-low pulse. Active-low acknowledges from all neurons in the same row are NANDed together to generate a active-high row acknowledge ryi. These row-acknowledges are NORed together to produce a single active-low acknowledge that is sent back to the decoders.

We can use the same control circuit shown in Fig. 13(b) to interface a column of neurons with the arbiter and the encoder. The column logic itself consists of a Y-input wired-NOR gate, which feeds c̃ox into p̃, and Y *aC-elements*, which steer s to the correct neuron by ANDing it with the row-select. We can eliminate the *aC-elements* and broadcast the controller's acknowledge, since it is already ANDed with the row-select signal inside the neuron, provided we clear it before a new row is selected.

To figure out if a new row can be selected before the column-select is cleared, note that the row-controller selects a new row *after* the row-encoder clears its acknowledge (i.e., [ãi]), and this signal is essentially synchronous with the column-encoder's acknowledge, as the encoders simply relay the receiver's acknowledge. Furthermore, the column- (and row-) controller's select signals must be low in order for the encoders to clear their requests to the receiver. Hence, it follows that the column-select signal is cleared before a new row is selected.

Throughput may be boosted by reading the state of all neurons in a selected row in parallel, and storing their spikes in a latch on the periphery of the array, where they can be rapidly relayed to the column arbiter. Stored spikes are transmitted in a rapid burst, while the array is cycled to select and read the next row. The performance enhancement achieved by this approach is described briefly in Section V; design details may be found in [48]. Let us now turn our attention to organizing the receiving neurons into rows and columns.

2) 2-D Receiver: The 2-D AER receiver's structure parallels that of the transmitter, as shown in Fig. 12. First, we use a $\lceil \log_2(Y) \rceil$-bit decoder to select one of the Y rows, and then we use a $\lceil \log_2(X) \rceil$-bit decoder to select one of the X output ports assigned to that row.

This strategy is realized by the circuit shown in Fig. 14, obtained by decomposing AERCV(N) into neuron, row, and column subprocesses, and following the synthesis procedure. The gate that combines the row- and column-selects is changed from a state-holding C-element to a purely combinational NAND gate. Thus, we clear our request to the neuron when either the column select or the row select is cleared, without waiting for the other line to clear. We run the risk of choosing the wrong neuron next time around, when a lingering row select signal is NANDed with a new column select signal, or vice versa. Matching the decoders' delays minimizes the risk. Asynchronous versions of traditional circuits used to encode and decode addresses are shown in Fig. 15.

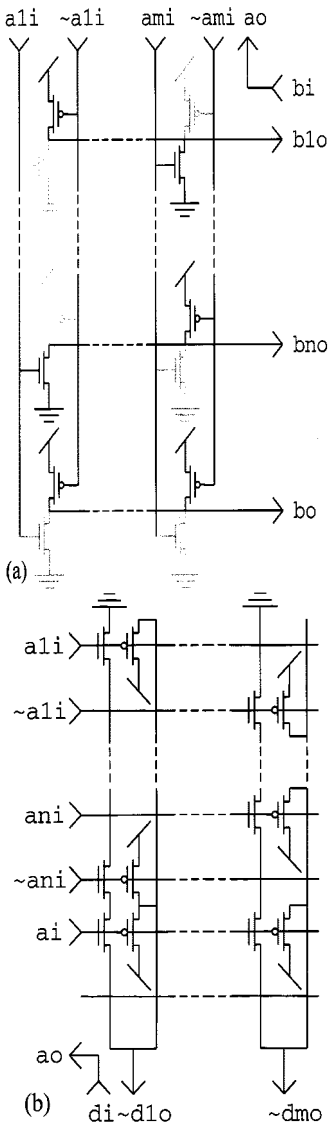


Fig. 15. Address-encoder and -decoder circuits. (a) A 1-in-m one-hot to n-bit binary encoder, where $n = \lceil \log_2(m) \rceil$ is built with $m \times (n+1)$ one-transistor cells. A cell either pulls up the output line with a pFET, driven by an active-low input line (i.e., ām̃i), or pulls it low with a nFET, driven by an active-high input line (ami). An extra output line (bi) that is always pulled high provides an active-high request signal. (b) A n-bit to 1-in-m decoder, where $m < 2^n$, is built with $(n+1) \times m$ two-transistor cells. The $n+1$ cells connected to each active-low output (i.e., d̄mo) form an $(n+1)$-input NAND gate, with $n+1$ series-connected nFET's and $n+1$ parallel-connected pFET's. We decode a zero or a one by driving the cell with either an active-high signal (ani) or an active-low signal (āni), respectively. The active-high request signal is connected to the $(n+1)$th input (adapted from [10]).

Our review of 2-D AER transmitter and receiver design is now complete. We have seen how to reduce the overhead imposed by arbitration, encoding, and decoding from N to $\sqrt{N}$ by organizing neurons into rows and columns, and how to exploit this organization, together with locality, to reduce the average cycle time. As shown in Fig. 12, and described in [13], [14], a pipeline stage [see Fig. 9(b)] can be inserted between

the receiver's input port and its decoders to improve its performance. This slack allows the receiver to acknowledge as soon as it latches the address from bus, and then decode the address and select the neuron while the sender is clearing it row or column select signals and selecting a new row or column. Having addressed the intricacies of asynchronous logic circuit design, we now turn our attention to the pitfalls of mixed-analog–digital design.

D. Analog–Digital Interfaces

Neuromorphic chips are MAD systems [49], where sending and receiving neurons serve as analog-to-digital and digital-to-analog converters. They use subthreshold analog CMOS circuits to model dendritic computation [3] and asynchronous digital CMOS logic to model axonal communication [3], [50]. One of the greatest difficulties in their design is reducing crosstalk between the analog and digital parts, given the gigantic differences in current levels and speeds.

In the analog domain, we use 100-pA currents and 100-fF capacitors to achieve 1-V/ms slew rates. Whereas in the digital domain 100-μA currents and 100 fF capacitors yield 1-V/ns slew rates, a million times higher! To match these slew rates, the neuron's gain must exceed one million. And to ensure that less than 5 mV of the 5-V digital swing finds its way into the analog circuitry, parasitic coupling capacitances must be less than 0.1 fF! Recalling that a CMOS inverter's gain is about 10 and a minimum-sized transistor's drain-to-gate overlap capacitance exceeds 1 fF, you realize how demanding these specifications are.

To realize a millionfold gain, we use a two-inverter noninverting amplifier with positive feedback, also known as the axon-hillock circuit. This circuit, shown in Fig. 13(a), is named after the spike-initiation zone in a biological neuron [3]. If the loop gain exceeds unity, the output's rate of change is limited only by the amplifier's output current, not by the input current. Thus, this circuit has an effective gain of 100 000 or more! Unfortunately, the first inverter, with its input charging up at 1 V/ms, spends 0.5 ms within 0.5 V of threshold, passing a short-circuit current close to 100 μA the whole time. Hence, it consumes a million times more energy than a regular CMOS inverter.

We may limit the axon-hillock's power dissipation by starving the first inverter using an nMOS-style pull-up transistor, which supplies a fixed bias current of about 1 μA, as suggested by Lazzaro [51]. It is unsafe to reduce the current further because this inverter's output must switch all the way to `Gnd` by the time the row is selected [see Fig. 13(a)]. Otherwise, the second inverter's pull-down transistor will clear `lix` when its pull-up is disabled by `s` going high.[4] Consequently, this approach reduces the power dissipation to only 10 000 times that of a regular inverter.

Power-supply rails mediate crosstalk. Transistors connected to the rails form a multiple-input differential pair, and a device transiently steals current from the others when it is switched on. With the axon-hillock's input and threshold transistors tied to

[4]Charles Higgins discovered this race condition.

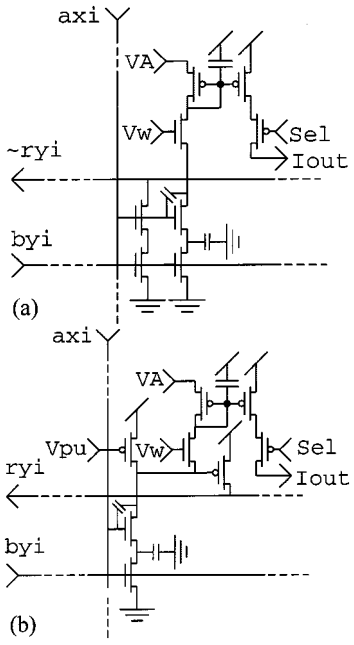


Fig. 16. Bad and good receiver pixels. Both pixels use a diode-capacitor circuit to integrate spikes and a tilted current-mirror to amplify current, as described in [32]. The bias voltage `Vw` sets the amount of charge metered onto the capacitor each time the pixel is selected. The parasitic capacitors shown can pump or inject current into the integrator, as explained in the text. In (b), the pull-up isolates the integrator from these parasitic effects.

`Gnd` and `Vdd`, respectively, these rails mediate inhibitory and excitatory interactions, respectively [see Fig. 13(a)]. We can avoid turning off the current in these transistors by limiting the reset current and turning it off as soon as the spike is reset, as described in [3]. Hence, we can isolate the neuron from the digital circuitry by moving this inverter to the analog supply, without corrupting the analog supply. However, we must also move the second inverter's pull-down to the analog supply to avoid injecting digital supply noise through the positive-feedback capacitor (via the second inverter's pull-down device, which remains on during the interspike interval). The inhibition mediated by this pull-down is acceptable, as it tends to desynchronize the neurons. Unlike excitation, which tends to synchronize the neurons and increase spiking activity.

Parasitic capacitances within a device, due to overlap between gate and source/drain diffusion, can turn on a device by driving its source outside the supply rail. This problem plagued the first receiver pixel designed [shown in Fig. 16(a)]. Rapid voltage swings on the column-select line (`axi`) are transmitted to the source terminal of the current-source transistor (device with gate tied to `Vw`), driving it a fraction of a volt below `GND`—if the node's voltage sits close to `GND`, as it does in this circuit. As a result, the current source would pass a fraction of a picoamp even when `Vw` was tied to `GND`.

Parasitic capacitances between series-connected devices can produce charge-pumping. This problem also plagued the receiver pixel shown in Fig. 16(a). The pair of transistors controlled by the row and column select lines, `byi` and `axi`, pump charge from the current-source transistor to ground when nonoverlapping pulses occur on the select lines. For a 20 fF parasitic capacitor, a 100-Hz average spike rate per pixel, a

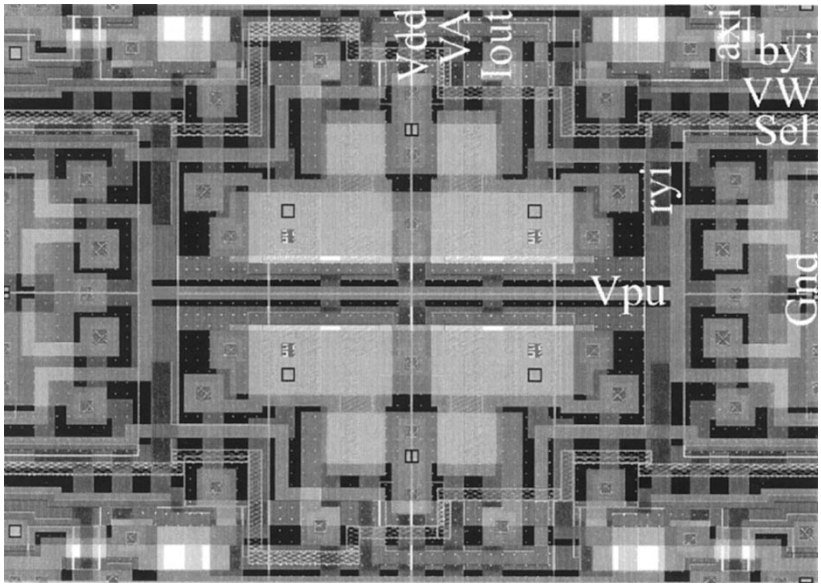


Fig. 17. Layout of 2 × 2 receiver pixels. Pixels are flipped vertically and horizontally to isolate digital and analog circuitry and share contacts [see Fig. 16(b) for the circuit]. Current-mirror integrators are located centrally, with switched current-sources (large devices at top and bottom edges), NAND-gate pull-downs (two small series-connected devices tied to the current-sources), and NAND-gate and acknowledge pull-ups (devices with L-shaped gates near left and right edges) on the periphery. `axi`, `ryi`, `VA`, and `Iout` run vertically in M1, `byi` runs horizontally in M2, `VW`, `Vpu` and `Sel` run horizontally in Poly1. A second Poly2 select line is used in odd columns to compensate for hexagonal tiling in the retina chip. An M1 line, tied to `Vdd`, shield's the current-source transistor's drain from the M2 row-select line. Gray is substrate; black and darker shading is M2. The cell width is 63λ and its height is 46λ, or $18.9\ \mu$m $\times$ $13.8\ \mu$m in 0.5-μm technology ($\lambda = 0.3\ \mu$m).

0.5-V voltage drop, and 64 neurons per row or column, the current is 64 pA. This current, which scales with the array size, easily swamps out the subpicoamp current levels we must maintain in the diode-capacitor integrator to obtain time constants greater than 10 ms using a 300-fF capacitor.

Capacitive turn-on and charge-pumping can both be eliminated by adding a pull-up, which implements an nMOS-style NAND gate, as shown in Fig. 16(b). A full CMOS NAND gate will also work—it eliminates the global bias line `Vpu` but requires an additional transistor. The pull-up keeps the current-source transistor's source-terminal close to `Vdd`, making it impossible to capacitively drive it below ground, and it is biased to supply a few microamps, easily overwhelming the pump current. Furthermore, the current-source transistor is switched on by swinging its source terminal from `Vdd` to `GND`, a technique that can meter minute quantities of charge, as demonstrated by Cauwenberghs [52]. A layout of the receiver pixel is shown in Fig. 17. In the next section, characterization procedures for AER communication channels and the performance of some existing designs are discussed and reviewed, respectively.

V. Test Protocols and Channel Performance

Timing relationships between the control signals must be kept in mind while debugging these asynchronous interfaces. The diagram in Fig. 7(b) indicates which party must act at any stage in the sequence, helping us to determine who is at fault when the channel hangs. For example, if it hangs with both `r` and `a` high, the arrow indicates that the sender is at fault; it failed to withdraw its request. Testing is facilitated by interfacing sender and receiver chips with a computer that can read and write addresses at high speed. Delbrüuck *et al.* have implemented a MatLab-based interface on the Mac, using a parallel I/O card from National Instruments [53]. They achieved a transfer rate of 100 kHz by programming at the register level.

TABLE IV
Three Generations of Arbitered Channels

Size	Process	Read-Out	Cycle-Time	Throughput
64×64	2.0μm	Random	2μs	500KS/s [4]
64×64	2.0μm	Local	420-730ns	2.0MS/s [13]
104×96	1.2μm	Parallel	30-400ns	25MS/s [48]

The architectural optimizations described earlier reduced cycle times by more than an order of magnitude, over three generations of arbitered AER channel designs. Going from 2 μs, reported in Mahowald and Sivilotti's pioneering work, to as low as 30 ns reported in [48], where spikes are readout from the array in parallel. Table IV summarizes the evolution.

Address-event streams from the local-readout 64×64 neuron transmitter design [14], fabricated in 2-μm technology, reveal the arbiters' greedy behavior. This transmitter uses the architecture shown in Fig. 12, and reads all the spiking neurons in a selected row, sequentially, before it selects another row. The row arbiter rearranges the Y address, as shown in Fig. 18, as it attempts to span the smallest subtree, going to the nearest row that is active. Such scanning is beneficial because transversing an additional level added 37 ns (estimated) to the cycle time. Scanning is not evident in the X addresses because, at the low load level used, no more than three or four neurons are active simultaneously within the same row.

Cycle-time measurements from a parallel-readout 104×96 neuron transmitter [48], fabricated in 1.2-μm CMOS technology, are shown in Fig. 19. This transmitter used an architecture similar to the previous one, except that a row-wide latch was interspersed between the array and the column arbiter, and the state of all neurons in a selected row were

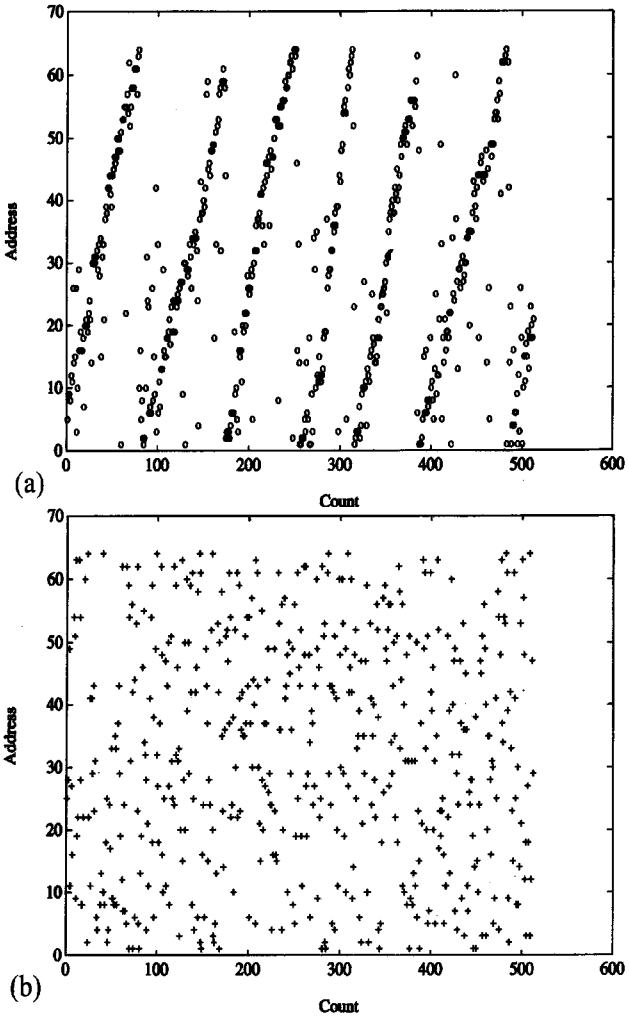


Fig. 18. Address-event streams showing arbiter scanning. Y and X addresses are plotted on the vertical axes, and their position in the stream is plotted on the horizontal axes. For load at 5% capacity: (a) Y addresses tend to increase with sequence number; and (b) X addresses are distributed randomly.

read in parallel. The latch's bit-cells act as slave neurons: they send requests to the column arbiter, while the neurons in the selected row are reset and another row is selected. The cycle-time is as low as 30 ns when spikes are read from the latch, and goes up to 400 ns when data is read from the array. Even this worst-case cycle-time, which involves arbitration in both dimensions, trumps the 730 ns achieved with the earlier local-readout design [13], [14].

Images in Fig. 20 show the response of the 104×96 neuron retinomorphic chip to a light spot [31]. The address events were read into a computer, and the images were rendered by modeling temporal integration in the diode-capacitor integrator. The four types of ganglion cells on the chip subsample the image by a factor of two; LSB's of the X and Y addresses encode the cell type.

VI. Discussion and Summary

This paper has described the design of communication channels for neuromorphic chips. These designs exploit spatially sparse, temporally coincident, neural activity, whitened by preprocessing in the sensory epithelium. Neuronal ensembles capture this stimulus-driven spatiotemporal activity. They consists of spikes clustered at distinct temporal locations where events occur and sparse spatial locations determined by the stimulus pattern. They also have an unstructured component that arises from noise in the signal, noise in the system, and from differences in gain and state among the neurons. This stochastic component limits the precision with which neurons encode information about the stimulus. Neuronal ensembles can be transmitted by time-division multiplexing without losing information if the channel's timing precision exceeds that of the neurons.

A. Design Tradeoffs

For random-access time-multiplexed channels, the multiple bits required to encode identity are offset by the reduced sampling rates produced by local adaptation when activity is sparse. The payoff is even better when there are sufficient I/O pins to transmit all the address-bits in parallel. In this case, frequency and time-constant adaptation allocate bandwidth dynamically in the ratio a: $(1-a)/Z$ between active and passive fractions of the population. For low active fractions a and sampling-rate attenuation factors Z larger than $1/a$, the effective Nyquist sampling rate may be increased by a factor of $1/2a$.

Contention occurs when two or more neurons spike simultaneously, and we must dump old spikes to preserve the timing of new spikes or queue new spikes to avoid losing old spikes. An unfettered design, which discards spikes clobbered by collisions, offers higher throughput if high spike-loss rates are tolerable. In contrast, an arbitered design, which makes neurons wait their turn, offers higher throughput when low-spike loss rates are desired. Indeed, the unfettered channel utilizes only 18% of its capacity at the most. Therefore, the arbitered design offers more throughput if its cycle time is no more than five times longer than that of the unfettered channel.

The inefficiency of the unfettered channel design, also known as ALOHA, has been long recognized, and more efficient protocols have been developed [7]. One popular approach is carrier sense multiple access (CSMA), where each user monitors the channel and does not transmit if it is busy.[5] This channel is prone to collisions only during the time it takes to update its state. Hence, the collision rate drops if the round-trip delay is much shorter than the packet-transmission time, as in bit-serial transmission of several bytes. Its performance is no better than ALOHA's; however, if the round-trip delay is comparable to the packet-transmission time [7], as in bit-parallel transmission of one or two bytes. Consequently, it is unlikely that CSMA will prove useful for neuromorphic systems (preliminary results are reported in [54]).

As technology improves and we build denser arrays with shorter cycle times, the unfettered channel's collision probability remains unchanged for the same normalized load, whereas the arbitered channel's normalized timing error decreases. This desirable scaling arises because timing error is the product of the number of wait cycles and the cycle-time. Consequently, queuing time decreases due to the shorter cycle

[5]The Ethernet, and most local-area-networks, work this way.

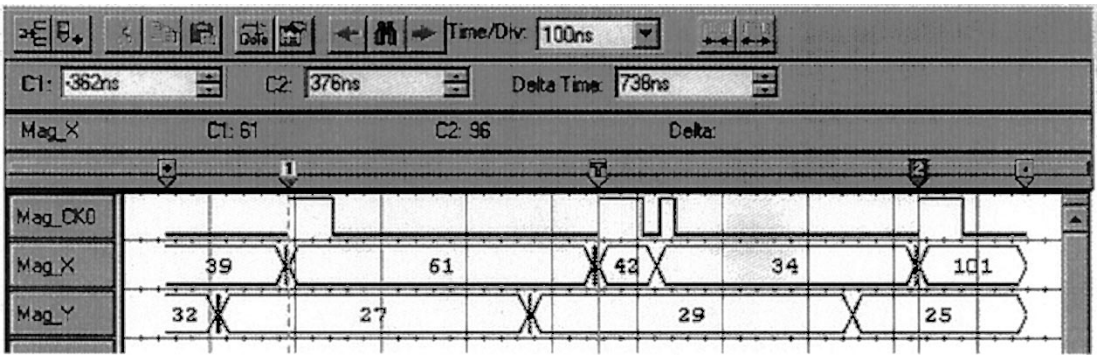


Fig. 19. Cycle times for parallel-readout transmitter—screen dump from a Tektronix TLA704 Logic Analyzer. Top trace: request. Middle: column address. Bottom: row address. The first and second address events are from different rows, whereas the second and third events are from the same row. The cycle time is 362 ns in the first case and 72 ns in the second case.

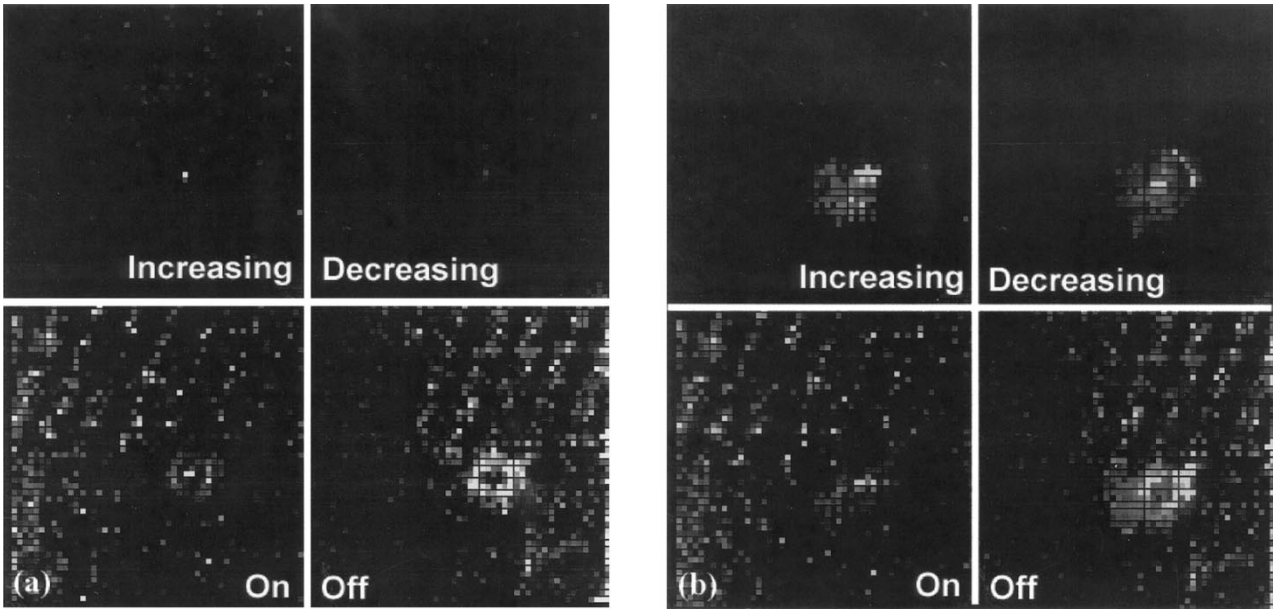


Fig. 20. Response of retinomorphic chip. Four ganglion-cell types respond to light or dark spots either in a transient (Increasing, Decreasing) or sustained fashion (On, Off). (a) Light spot stationary (located where the single active Increasing cell is): sustained cells pick up increased signal at the spot's location and decreased signal in surrounding region, due to lateral inhibition. Fixed-pattern noise, due to transistor mismatch, is also evident. (b) Light spot moving up and to the right: transient cells pick up decrease at inhibitory surround's leading edge and increase at excitatory center's leading edge. The mean spike rate was 5 spikes/neuron/s.

times, even though the number of cycles spent waiting remains the same. Indeed, as the cycle-time must be inversely proportional to the number of neurons N, the normalized timing error is less than $400/N$ for loads below 95% capacity and active fractions above 5%. For population sizes of several tens of thousands, the timing errors is just a few percentage points.

For neurons whose timing precision is much better than their interspike interval, we may estimate throughput requirements by measuring frequency adaptation and synchronicity. Frequency adaptation γ gives the spike rate for neurons that are not part of the neuronal ensemble, and synchronicity ξ gives the peak spike rate for neurons in the ensemble. These firing rates are obtained from the spike frequency at stimulus onset by dividing by γ and multiplying by ξ, respectively. The throughput must exceed the sum of these two rates if we wish to transmit the ensemble without adding latency or temporal dispersion. The surplus capacity must be at least 455% to account for collisions in the unfettered channel, but may be as low as 5.3% in the arbitered channel, with subpercent timing errors due to queuing.

B. Asynchronous Implementation

$N - 1$ two-input arbiter cells, arranged in a tree with $\lfloor \log_2(N) \rfloor$ levels, are required to arbitrate between N neurons, plus N address decoder and encoder cells. This area overhead may be reduced from N to $\sqrt{N}$ by going to a hierarchical row–column organization. Time overhead is reduced by adopting three strategies.

Pipelining reduces the time overhead of arbitration by overlapping communication sequences of the sending neuron, the row arbiter, the column arbiter, and the receiving neuron. We also inserted a pipeline stage between the receiver chip's input port and its decoders, allowing it to acknowledge as soon as it latches the address from bus. It decodes the address and selects the target neuron, while the sender is clearing its row or column select signals and selecting a new row or column.

Exploiting locality in the row–column organization reduces time-overhead further by servicing all active neurons in the selected row, redoing row arbitration only when no requests are left. Throughput is boosted further by reading the state of all neurons in a selected row in parallel, storing this information in a latch at the periphery of the array, where it can be readily supplied to the column arbiter. The spikes are transmitted in a rapid burst, while the array is cycled to select and read the next row.

Exploiting locality in the arbiter tree also reduces time overhead by spanning the smallest subtree that has a pair of active inputs. We implemented this strategy simply by making the two-input arbiter cell greedy; it services both of its daughters if they are active. Thus, the arbiter becomes a scanner when neuronal activity is spatially clustered. It transverses one level with probability 1/2, two with probability 1/4, three with probability 1/8, and so on; this series converges to two levels.

However, exploiting locality trades fairness for efficiency. Instead of allowing every active neuron to bid for the next cycle, or granting service on a strictly first-come–first-served basis, the transmitter acts like a traveling salesman and services the closest customer. It may pick up a neuron that is close by over neurons that fired earlier. Giving priority to location minimizes the average cycle time, thereby maximizing the channel capacity and minimizing the average wait time. Unfortunately, service is limited to a local area when the channel is overloaded, and neurons outside that area are simply ignored. Irrespective of the fairness of the selection mechanism we choose, the average wait time goes to infinity when the channel capacity is exceeded. Therefore, maximizing channel capacity is my paramount concern.

Special attention must be paid to digital–analog interfaces in these hybrid designs. It is imperative to use positive feedback to match analog and digital slew rates; hysteresis alone [41] is not enough. However, the axon-hillock circuit's power dissipation must be reduced drastically (possibly by adapting the bias current to the output's rate of change) and excitatory coupling through the supply rails carefully isolated. Capacitive turn-on and charge-pumping in receiver-neuron interfaces were eliminated by implementing an nMOS-style NAND gate.

We may have to adopt bidirectional current-signaling to achieve subpercent ($<$40 dB) capacitive crosstalk. However, this technique, which has been used within [55], [24] and between [56]–[58] chips, must be refined further to reduce static power dissipation, and the area-overhead in capacitors for frequency-compensation or in large devices for better matching.

Following a rigorous design methodology for asynchronous logic circuits has paid off, improving robustness and reliability considerably. Although the state-of-the-art implementations are by no means bullet-proof [48], they are getting to the point where nonexperts can use them with the aid of silicon compilation. Making it possible for neuromorphic-system designers with limited expertise in asynchronous communication to successfully incorporate these channels into their designs [17], [59].

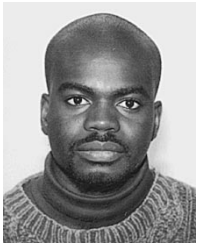
ACKNOWLEDGMENT

The author would like to thank C. Mead for sharing insights into nervous system organization. The author would also like to thank M. Mahowald for making available layouts of the arbiter, the address encoders, and the address decoders; J. Lazzaro, A. Martin, J. Tierno, and T. (Bassen) Lande for helpful discussions; T. Delbrück for help with the Macintosh AER interface and L-Comp; and J. Dickson for help with PCB design, and Tanner Research Inc. for making available a prerelease version of their L-Comp libraries for silicon compilation.

REFERENCES

[1] C. A. Mead, "Neuromorphic electronic systems," *Proc. IEEE*, vol. 78, pp. 1629–1636, Oct. 1990.
[2] E. A. Vittoz, "Analog VLSI Implementation of Neural Networks," in *Handbook of Neural Computation*, E. Fiesler and R. Beale, Eds. Oxford, U.K.: Oxford Univ. Press, 1995.
[3] C. A. Mead, *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley, 1989.
[4] M. Mahowald, "VLSI analogs of neuronal visual processing: A synthesis of form and function," Ph.D. dissertation, California Inst. Technol., Pasadena, CA, 1992.
[5] K. A. Boahen, "Retinomorphic vision systems: Reverse engineering the vertebrate retina," Ph.D. dissertation, California Inst. Technol., Pasadena, CA, 1997.
[6] T. S. Lande, Ed., *Neuromorphic Systems Engineering: Neural Networks in Silicon*. Norwell, MA: Kluwer, 1998.
[7] M. Schwartz, *Telecommunication Networks: Protocols, Modeling, and Analysis*. Reading, MA: Addison-Wesley, 1987.
[8] A. S. Tanenbaum, *Computer Networks*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1989.
[9] M. Sivilotti, "Wiring considerations in analog VLSI systems, with application to field-programmable networks," Ph.D. dissertation, California Inst. Technol., Pasadena, CA, 1991.
[10] M. Mahowald, *An Analog VLSI Stereoscopic Vision System*. Norwell, MA: Kluwer, 1994.
[11] T. Sejnowski, C. Koch, and R. Douglas, Eds., (1997) Telluride Workshop on Neuromorphic Engineering. [Online]. Available: http://www.klab.caltech.edu/ harrison/tell97/report97/index.html
[12] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti, and D. Gillespie, "Silicon auditory processors as computer peripherals," *IEEE Trans. Neural Networks*, vol. 4, pp. 523–528, 1993.
[13] K. A. Boahen, "Retinomorphic vision systems ii: Communication channel design," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 1996, pp. 14–17.
[14] ——, "Communicating neuronal ensembles between neuromorphic chips," in *Neuromorphic Systems Engineering: Neural Networks in Silicon*, T. S. Lande, Ed. Norwell, MA: Kluwer, 1998, ch. 11.
[15] S. R. Deiss, R. J. Douglas, and A. M. Whatley, "A pulse-coded communications infrastructure for neuromorphic systems," in *Pulsed Neural Networks*, W. Maass and C. M. Bishop, Eds. Cambridge, MA: MIT Press, 1999, ch. 6, pp. 157–178.
[16] J. P. Lazzaro and J. Wawrzynek, "A multi-sender asynchronous extension to the address-event protocol," in *Proc. 16th Conf. Advanced Research in VLSI*, W. J. Dally, J. W. Poulton, and A. T. Ishii, Eds., 1995, pp. 158–169.
[17] C. M. Higgins and C. Koch, "Multi-chip motion processing," in *Proc. 20th Anniversary Conf. Advanced Research in VLSI*, D. S. Wills and S. P. DeWeerth, Eds., 1999.
[18] S. P. DeWeerth, G. N. Patel, M. F. Simoni, D. E. Schimmel, and R. L. Calabrese, "A vlsi architecture for modeling intersegmental coordination," in *Proc. 17th Conf. Advanced Research in VLSI*, R. Brown and A. Ishii, Eds., 1997, pp. 182–200.
[19] K. A. Boahen, A. G. Andreou, T. Hinck, J. Kramer, and A. Whatley. (1997.) *Computation and memory-based projective field processors*. Telluride Workshop on Neuromorphic Engineering. [Online]. Available: http://www.klab.caltech.edu/ harrison/tell97/report97/index.html
[20] J. G. Elias, "Artificial dendritic trees," *Neural Comput.*, vol. 5, pp. 648–663, 1993.
[21] K. A. Boahen, "Retinomorphic vision systems," in *Proc. Microneuro'96: 5th Int. Conf. Neural Networks and Fuzzy Systems*, Feb 1996, pp. 2–14.
[22] ——, "A retinomorphic vision system," *IEEE Micro*, vol. 16, pp. 30–39, Oct. 1996.
[23] M. Mahowald and C. A. Mead, "The silicon retina," *Sci. Amer.*, vol. 264, no. 5, pp. 76–82, 1991.

[24] K. A. Boahen and A. Andreou, "A contrast-sensitive retina with reciprocal synapses," in *Advances in Neural Information Processing 4*, J. E. Moody, Ed. San Mateo, CA: Morgan Kauffman, 1992, vol. 4, pp. 764–772.
[25] R. F. Lyon and C. A. Mead, "An analog electronic cochlea," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 1119–1134, 1988.
[26] L. Watts, "Cochlear mechanics: Analysis and analog VLSI," Ph.D. dissertation, California Inst. Technol., Pasadena, CA, 1993.
[27] M. V. Srinivasan, S. B. Laughlin, and A. Dubs, "Predictive coding: A fresh view of inhibition in the retina," in *Proc. R. Soc. Lond. B Biol. Sci.*, vol. ASSP-216, 1982, pp. 427–459.
[28] J. Atick and N. Redlich, "What does the retina know about natural scene," *Neural Comput.*, vol. 4, no. 2, pp. 196–210, 1992.
[29] D. K. Warland, P. Reinagel, and M. Meister, "Decoding visual information from a population of retinal ganglion cells," *J. Neurophys.*, vol. 78, pp. 2336–2350, 1997.
[30] T. Delbruck and C. A. Mead, "Analog VLSI Phototransduction by Continuous-Time, Adaptive, Logarithmic Phtoreceptor Circuits," Computation and Neural Systems Dept., California Inst. of Technol., Pasadena, CA, CNS Memo #30, 1994.
[31] K. A. Boahen, "Retinomorphic chips that see quadruple images," *Proc. Microneuro'99: 7th Int. Conf. Neural, Fuzzy, and Bio-Inspired Systems*, pp. 12–20, Apr. 1996.
[32] ——, "The retinomorphic approach: Pixel-parallel adaptive amplification, filtering, and quantization," *Analog Integ. Circuits Signal Processing*, vol. 13, pp. 53–68, 1997.
[33] R. Etienne-Cummings, J. Van der Spiegel, P. Mueller, and M. Zhang, "A foveated silicon retina for two-dimensional tracking," IEEE Trans. Circuits Syst. II, to be published.
[34] D. N. Mastronade, "Conrrelated firing of cat retinal ganglion cells—I: Spontaneuosly active inouts to x- and y-cells," *J. Neurophysiol.*, vol. 49, pp. 303–324, 1983.
[35] M. Meister, L. Lagnado, and D. A. Baylor, "Concerted signaling by retinal ganglion cells," *Science*, vol. 270, pp. 1207–1210, 1995.
[36] W. M. Usrey, J. B. Reppas, and R. C. Reid, "Paired-spike interactions and synaptic efficacy of retinal inputs to the thalamus," *Nature*, vol. 395, no. 6700, pp. 384–387, 1998.
[37] A. Murray and L. Tarassenko, *Analogue Neural VLSI: A Pulse Stream Approach*. London, U.K.: Chapman and Hall, 1994.
[38] L. M. Reyneri, "A performance analysis of pulse stream neural and fuzzy computing systems," *IEEE Trans. Circuits Syst. II*, vol. 42, pp. 642–40, Oct. 1995.
[39] W. R. Softky, "The highly irregular firing of cortical cells is inconsistent with temporal integration of random epsps," *J. Neurosci.*, vol. 13, pp. 334–350, 1993.
[40] W. Maass and C. M. Bishop, Eds., . Cambridge, MA: MIT Press, 1999.
[41] A. Mortara, E. Vittoz, and P. Venier, "A communication scheme for analog vlsi perceptive systems," *IEEE Trans. Solid-State Circuits.*, vol. 30, pp. 660–669, June 1995.
[42] L. Kleinrock, *Queueing Systems*. New York, NY: Wiley, 1976.
[43] A. J. Martin, S. M. Burns, T. K. Lee, D. Borkovic, and P. J. Hazewindus, "The design of an asynchronous microproessor," in *Proc. Decennial Caltech Conf.: Advanced Research in VLSI*, C. L. Seitz, Ed., 1989, pp. 351–373.
[44] I. E. Sutherland, "Micropipelines," *Commun. ACM*, vol. 32, no. 6, pp. 720–738, 1989.
[45] A. Martin, "Programming in vlsi: From communicating processes to delay-insensitive circuits," California Inst. Technol., Pasadena, CA, Tech. Rep. CS-TR-89-01, 1989.
[46] A. J. Martin, "Compiling communicating processes into delay-insensitive vlsi circuits," *Distrib. Comput.*, vol. 1, no. 4, pp. 226–234, 1990.
[47] A. J. Martin, "Asynchronous datapaths and the design of an asynchronous adder," *Formal Meth. Syst. Design*, vol. 1, no. 1, pp. 119–137, 1990.
[48] K. A. Boahen, "A throughput-on-demand address-event transmitter for neuromorphic chips," in *Proc. 20th Aniversary Conf. Advanced Research in VLSI*, D. S. Wills and S. P. DeWeerth, Eds., 1999, pp. 72–86.
[49] J. Franca and Y. Tsividis, *Design of Analog–Digital Vlsi Circuits for Telecommunications and Signal Processing*. Englewood-Cliffs, NJ: Prentice-Hall, 1994.
[50] B. A. Minch, P. Hasler, C. Diorio, and C. A. Mead, "A silicon axon," in *Advances in Neural Information Processing Systems 7*, G. Tesauro, D. S. Touretzky, and T. K. Leen, Eds. Cambridge, MA: MIT Press, 1995, pp. 739–746.
[51] J. P. Lazzaro, "Low-power silicon spiking neurons and axons," in *Proc. IEEE Int. Symp. Circuits and Systems*, 1992, pp. 2220–2224.
[52] G. Cauwenberghs and A. Yariv, "Fault-tolerant dynamic multi-level storage in analog vlsi," *IEEE Trans. Circuits Syst. II*, vol. 41, pp. 827–829, Dec. 1995.
[53] T. Delbruck, H. Floberg, and L. Peterson. (1994) Address-event communication using lab-nb board, matlab, and a mac. California Institute of Technology, Pasadena, CA. [Online]. Available: http://www.pcmp.caltech.edu/aer/txrx/txrx.pdf
[54] A. Abusland, T. S. Lande, and M. Hovin, "A vlsi communication architecture for stochastically pulse-encoded analog signals," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 1996, vol. III, pp. 401–404.
[55] K. A. Boahen, P. O. Pouliquen, A. G. Andreou, and A. Pavasovic, "Architectures for associative memories using current-mode mos circuits," in *Proc. Decennial Caltech Conf. VLSI*, C. L. Seitz, Ed., 1989, pp. 175–193.
[56] K. Lam, L. Dennison, and W. Dally, "Simultaneuos bidirectional signalling for ic systems," in *Proc. 1990 Conf. Computer Design (ICCD)*, 1990, pp. 430–433.
[57] L. Dennison, W. Lee, and W. Dally, "High-perfomance bidirectional signalling in vlsi systems," in *Proc. 1993 Symp. Research on Integrated Systems*, 1993, pp. 300–319.
[58] W. J. Dally and J. W. Poulton, *Digital Systems Engineering*. Cambridge, U.K.: Cambridge Univ. Press, 1998.
[59] G. A. Indiveri, M. Whatley, and J. Kramer, "A reconfigurable neuromorphic vlsi multi-chip system applied to visual motion computation," in *Proc. Microneuro'99: 7th Int. Conf. Neural, Fuzzy, and Bio-Inspired Systems*, 1999, pp. 37–44.



Kwabena A. Boahen received the B.S. and M.S.E. degrees in electrical and computer engineering from Johns Hopkins University, Baltimore, MD, in the concurrent Masters–Bachelors program, in 1989. He received the Ph.D. degree in computation and neural systems from the California Institute of Technology, Pasadena, CA in 1997, where he held a Sloan Fellowship for Theoretical Neurobiology.

He is an Assistant Professor in the Bioengineering Department, University of Pennsylvania, Philadelphia PA, where he holds a Skirkanich Term Junior Chair and a secondary appointment in electrical engineering. His current research interests include mixed-mode multichip VLSI models of biological sensory systems and asynchronous digital interfaces for interchip connectivity.

Dr. Boahen is a member of Tau Beta Kappa.